מחשבת

מאחורי הקלעים של תצוגת המסך

ATARI 9

600XL 800XL ATARI 400 and 800

ATARI
600XL, 800XL, 400, 800

מחשבת

מאחורי הקלעים
של תצוגת המסך

יחידה 9

מתווה התצוגה
(DISPLAY LIST)

תוכן העניינים

הקדמת המחבר

כילד (וגם כבוגר) חוויית השימוש שלי במחשב כללה כמובן משחק באינספור משחקי הפעולה המאתגרים והמרהיבים ביופיים שיש לאטארי להציע. במקביל, כבר מגיל צעיר התחלתי לחקור את יכולותיו של המחשב בעזרת כלי התכנות הזמינים כל כך בשפת הבייסיק.

והנה, ככל שהתקדמתי בלימוד רזי התכנות בבייסיק, כך הדהימה אותי רמת מורכבותם של משחקי המחשב (אשר רובם ככולם אינם כתובים בבייסיק). כיצד זה אפשרי, שאלתי את עצמי, במסגרת מגבלות החומרה, להפוך את מסך המחשב לתמונה מורכבת ועשירה כל כך, ואף מתנועעת באופן מהיר וחלק? כיצד למשל ניתן לשלב במסך אחד את הגרפיקה העדינה והמורכבת ביותר, ועדיין להשתמש בצבעים מרובים? – דבר שחורג לכאורה ממגבלות החומרה!

אקדים ואומר כי מענה לשאלות מעין אלו, מחייב שינוי של קו המחשבה, שינוי אשר ניתן לסכמו כך: אמנם אי אפשר לחרוג ממגבלות החומרה אך אפשר בהחלט לנצל את אופן פעולתה של החומרה, על מנת למקסם את יכולותיה.

אפשר להמשיל את העניין לרכיבה לילית באופניים. אמנם גלגל האופניים כשלעצמו (החומרה) אינו מסוגל להאיר לך את הדרך החשוכה, אך תוכל לנצל את סיבובו (אופן פעולת החומרה) ולהפיק ממנו זרם חשמלי המספיק להפעלת פנס קטן – וזאת אם תצמיד אל גלגל האופניים דינמו.

כמובן שכדי לנצל את יכולותיה של החומרה לצרכינו התכנותיים, עלינו להכיר היטב את אופן פעולתה. בכך אתמקד בחוברת לימוד זו.

נדב שטכמן פולישוק

אפריל 2021

מבוא

האם שאלת את עצמך אי-פעם, בשעה שהפעלת את המחשב שלך, או כאשר פקדת עליו: **GRAPHICS 0**, מה מתרחש ברקע? אילו פעולות הכנה "מאחורי הקלעים" נדרשות כדי שיתקבל אותו מסך טורקיז מוכר ובראשו המילה: READY

ביחידת לימוד זו נסיט הצידה את הקלעים, ונתבונן פנימה על מנת להבין מה מתחולל שם, בתוך השבב (המיקרו-מעבד) הגרפי של מחשב האטארי שלך. נלמד כיצד "יודע" אותו שבב – Antic שמו – לייצר את כל אחד מ-16 המסכים הגרפיים השונים המוכרים לך, בהתאם להנחיות הניתנות לו.

אחרי שנבין את הלוגיקה (הפשוטה למדיי) העומדת מאחורי ההנחיות הללו נוכל להשתמש בה כדי ליצור בעצמנו מסכים גרפיים חדשים ומורכבים יותר, המותאמים לצרכינו השונים. זו ואף זו – נלמד גם לנצל את הלוגיקה הזו כדי לייצר אפקט תנועה של המסך, אפקט הנקרא: גלילה.

בחלקה השני של החוברת נצלול עוד קצת לעומק ונחקור כיצד זורם הלכה למעשה המידע בין ה-Antic של המחשב לבין מסך הטלוויזיה שלך. בסיומה של החוברת נלמד לשלוט באותו "צינור" המזרים את המידע הגרפי ולהפעיל עליו מניפולציות שונות כדי ליצור על מסך הטלוויזיה שלך אפקטים מרהיבים.

אך רגע לפני שנצא אל המסע, ראוי שנצטייד בכלים הדרושים! ובכן, בפרק הראשון נסתפק בכלי התכנות של שפת הבייסיק. אך החל מן הפרק השני לא יהיה מנוס מלהכניס לשימוש גם את כלי התכנות של שפת אסמבלי. לכן, אם טרם עיינת במחשבת 8 "צעדים ראשונים בתכנות באסמבלי", מוטב שתעשה זאת כעת.

פרק א
מתווה התצוגה
(DISPLAY LIST)

דמה בנפשך כי אתה אדריכל העובד בשיתוף פעולה עם קבלן-בנייה. הקבלן מבצע את הנחיותיך, ואילו אתה מתבקש לשרטט עבורו מתווה תצוגה, מעין תוכנית מתאר לבניית מסך גרפי מספר 0.

? כיצד עשוי להיראות מתווה תצוגה שכזה?

התבונן היטב על המסך. בחלקו העליון ישנו שטח שחור שאינו מאפשר הקלדה ומתחתיו מתחיל המלבן הטורקיז, המאפשר להקליד 24 שורות של טקסט. אם תמדוד בסרגל את גובהו של האזור השחור העליון, תמצא שהוא שווה בגודלו לשלוש שורות טקסט. כלומר, סך כל היחידות האופקיות (הפעילות והלא פעילות) במסך גרפי 0 עומד על 27.

ואולם, על מנת להגיש מתווה תצוגה שלם, לא די בהגשת המפרט על אודות 27 יחידות אופקיות אלו, אלא שעליך לספק לקבלן שלך עוד שני פרטי מידע חשובים:

ראשית, עליך להודיע לו להיכן עליו לגשת כדי לקחת את כל אבני הבנייה הנדרשים לו (כלומר, היכן שמורים כל הסימנים הגרפיים הרשומים על המסך), ושנית, מה עליו לעשות **לאחר שסיים** את מלאכת הבנייה.

כאן המקום לספר לך כי קבלן-הבניין שלך הוא קבלן חרוץ עד מאוד. בניגוד לקבלן-בניין מסורתי המסיים את עבודת הבנייה והולך לנוח, הקבלן שלך אינו נח לרגע. לאחר שסיים לבנות (לפי הנחיותיך) את המסך הגרפי כולו, הוא ממהר לקפוץ אל נקודת ההתחלה, וממש כמו סיזיפוס המיתולוגי, מתחיל את כל תהליך הבנייה מחדש, שוב ושוב וחוזר חלילה.

וכל זאת למה? משום שמסך הטלוויזיה הוא דינאמי. טקסט שהיה כתוב עליו אך לפני רגע יכול להימחק ולהתחלף בטקסט אחר, וכדי להבטיח תנועה חלקה ורציפה, חייבת תצוגת המסך להתחלף שוב ושוב בקצב מסחרר.

? עד כמה מסחרר?

ובכן, קצב החזרה של קבלן הבנייה שלנו הוא 50 פעמים בשנייה, או בלשון "מחשבית" יותר: תדירותו היא 50 הרץ.[1]

כל זאת מוביל אותנו למסקנה הבאה: על מנת להשלים את מתווה התצוגה שלך ולהגישו לאותו קבלן-בניין חרוץ, עליך למסור לו מידע נוסף – מתי בדיוק עליו לבצע את אותה קפיצה "סיזיפית" אל נקודת ההתחלה, והיכן בדיוק נמצאת אותה נקודת התחלה.

אסכם: מתווה התצוגה שלנו צריך לכלול שלושה חלקים מרכזיים – ראשית, התווית מספרן של היחידות הגרפיות לסוגיהן (במקרה של מסך גרפי 0, ישנן כזכור 3 יחידות לא פעילות, ו-24 יחידות פעילות); שנית, ציון כתובת הזיכרון הראשונה בה שמור המידע הגרפי הממלא את המסך; ושלישית, הוראת

---

1 דע לך כי מחשבי אטארי המיועדים לשימוש בארצות הברית פועלים בתדירות גבוהה יותר, של 60 הרץ – תדירות המותאמת לשיטת השידור האמריקנית.

קפיצה הכוללת גם את ציון כתובת הזיכרון הראשונה בה שמור מתווה התצוגה עצמו.

הגיע הזמן לפרוש לפנינו את מתווה התצוגה הקיים עבור מסך גרפי 0 ולחקור אותו לעומקו. אך ראשית, עלינו כמובן למצוא אותו!

לשם כך משמשים זוג הבתים המצביעים – 560 ו-561. בבתים אלו שמורה כתובת זיכרון אשר מורה למחשב היכן יושב "מתווה התצוגה" – ה-Display List של המסך הגרפי הפעיל.

■ פקוד על המחשב להציג את ה-Display List של מסך 0.
דע לך כי מדובר במתווה תצוגה קטן למדי – 32 בתים גודלו.

[תשובה 1 בעמוד 105]

על המסך שלך אמורה להופיע עתה הרשימה הזו:

112,112,112,66,64,156,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,65,32,156

לצורך הדיון, נחלק אותה לחמישה מקטעים:

א	112	112	112																				
ב	66																						
ג	64	156																					
ד	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
ה	65	32	156																				

מקטע א' מורה ל-Antic למתוח שלוש יחידות אופקיות שחורות.

דע לך כי כל יחידה שכזו – המקודדת כערך 112 – בנויה למעשה מ-8 קווים אופקיים שחורים ברוחב של פיקסל, כל אחד.[2]

מקטע ב' בצירוף עם **מקטע ד'** מורים ל-Antic למתוח 24 יחידות אופקיות מסוג: "שורת טקסט".

שים לב להבדל בין המקטעים: מקטע ב' משמש מעין הצהרה. הערך 66 מורה ל-Antic כי לפניו Display List של גרפיקה מספר 0. אגב כך, נמתחת גם יחידת גרפיקה ראשונה על המסך, ממש מתחת לשטח השחור.

מקטע ד' לעומת זאת כולל רצף של 23 חזרות על הערך 2. ערכים אלו מורים ל-Antic למתוח 23 יחידות טקסט אופקיות נוספות, המשלימות ל-24 שורות הטקסט המוכרות לך ממסך גרפי 0.

לפני שנמשיך בחקירה שלנו, נעצור לרגע ונכניס מניפולציה קטנה ב-Display List שלנו.

■ אם עוד לא מחקת את התוכנית, פקוד על המחשב:

```
POKE DLIST+15,6: POKE DLIST+16,6
```

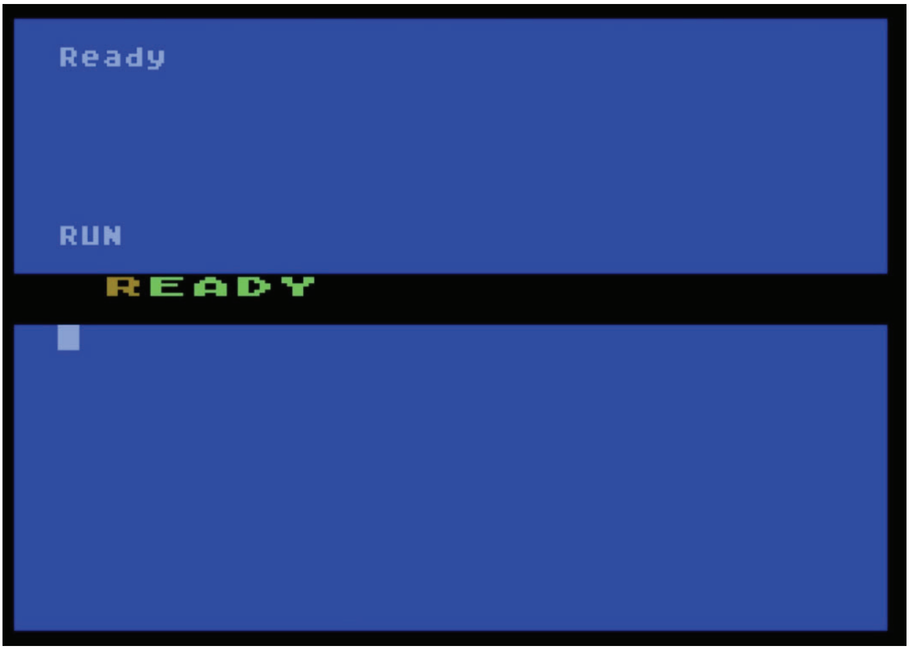
התבונן על מסך הטלוויזיה שלך. במרכז המסך נוצר חלון שחור ואם תקליד בתוכו תראה כי הטקסט מוגדל וצבעוני – ממש כמו טקסטים הנכתבים על מסך גרפי מספר 1.

אגב, כפי ששמת לב, שינינו ב-Display List שתי יחידות ולא הסתפקנו באחת, והסיבה לכך היא ההבדל באורכן של יחידות הטקסט השונות –

---

2 הסבר מעמיק על ערכי הקוד של מתווה התצוגה מופיע בנספח 1 להלן.

בעוד שאורכה של שורת טקסט במסך גרפי 0 הוא 40 תווים, אורכה של שורת טקסט במסך גרפי 1 הוא 20 תווים בלבד, וכדי להשלים את מניין 40 התווים, נדרשים עוד 20 תווים בטרם עוברים אל השורה הבאה.



- חזור על הפקודה שלעיל, אך הפעם במקום 6, הקלד את הערך 7 (בשתי הפקודות).

החלון השחור במרכז המסך גדל, כמו גם הטקסט הצבעוני הכתוב בתוכו – ממש כמו טקסט הנכתב על מסך גרפי מספר 2.

בשלב זה, תוכל כבר לשער כי ה-Display List של מסך גרפי 1, יכלול רצף של חזרות על הערך 6, ואילו ה-Display List של מסך גרפי 2 יכלול רצף של חזרות על הערך 7.

בהמשך אציג לוח המסכם את הערכים המקודדים את המסכים הגרפיים השונים, אך עתה נשוב ונחקור את המקטעים הנוספים ה-Display List של מסך גרפי 0.

מקטע ג' כולל את הערכים 156 ,64. זוג ערכים אלו משמש למעשה כבתים מצביעים (pointers) אשר מורים ל-Antic היכן נקודת ההתחלה של זיכרון המסך (כלומר: היכן שמורים כל הסימנים הגרפיים הממלאים את המסך).

כרגיל, כדי לחשב את כתובת הזיכרון נשתמש בשיטת ה-MSD, LSD , וכך:

156*256+64=40000

כתובת זיכרון זו אינה חדשה לך. כבר נתקלת בה בפרק א' של מחשבת 6, כשפגשת את שני הבתים המצביעים 88, 89.

בהמשך תבין על שום מה ולמה הכפילות, ומדוע נדרשת הפנייה לכתובת זיכרון המסך גם מתוך ה-Display List.

מקטע ה' פותח בערך 65. ערך זה מקודד פקודת דילוג. כאשר ה-Antic נתקל בפקודה הזו הוא יודע שעליו לדלג ולהמתין לריענון הבא של המסך. שני הערכים העוקבים, 32, 156 משמשים בתים מצביעים אשר מורים לו **להיכן** עליו לדלג.

גם הפעם, נשתמש בשיטת ה-MSD, LSD ונקבל:

156*256+32=39968

? האם מספר זה נראה לך מוכר?

בוודאי!

זוהי בדיוק כתובת הזיכרון של תחילת ה-Display List שחישבת קודם. לשם בדיוק מתבקש ה-Antic לקפוץ בחזרה ולחזור על בניית המסך כולו, שוב ושוב – 50 פעמים בשנייה.

לאחר שסיימנו לחקור את ה-Display List של מסך גרפי 0, הגיע הזמן לחקור Display List נוסף, מורכב יותר, ולצורך כך נעבור למסך גרפי 3.

חזור אל התוכנית הקצרה שכתבת, ושנה בשורה הראשונה את המסך הגרפי מ-0 ל-3.
שים לב, ה-Display List של מסך 3 מעט גדול יותר – בן 34 בתים.
כמו כן, קח בחשבון כי חלון הפלט של מסך גרפי 3 צר מאוד. על מנת להספיק להתבונן בתוצאה לפני שתיעלם, תוכל להוסיף לולאת השהייה.

על המסך שלך אמורה להופיע עתה רשימה של 34 ערכים. גם הפעם נחלק אותה לכמה מקטעים, אך באופן מעט שונה:

א	112	112	112																
ב1	72																		
ג1	112	158																	
ד1	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
ב2	66																		
ג2	96	159																	
ד2	2	2	2																
ה	65	78	158																

שים לב, ה-Display List מכיל שני רצפים: רצף ארוך של הערך 8, ועוד רצף קצר של הערך 2.

? האם יש לך מושג מדוע?

ובכן, המסך של גרפיקה 3 מורכב משני אזורים שונים באופיים – האזור העליון כולל 20 שורות של גרפיקה, ואילו האזור התחתון מכיל 4 שורות של טקסט (אותו חלון צר שמאפשר לך להקליד בתוכו).

ננתח את ה ה-Display List בפרוטרוט:

מקטע א' כולל, כרגיל, שלוש יחידות אופקיות שחורות.

מקטע ב1 כולל את הערך 72, אשר מצהיר כי לפנינו מסך גרפי 3 ואגב כך נמתחת יחידה אופקית ראשונה של גרפיקה על המסך.

מקטע ד1 ממשיך ומותח עוד 19 יחידות של גרפיקה (כדי להשלים ל-20 בסך הכול).

מקטע ב2, הכולל את הערך 66 (שכבר פגשנו ב-Display List הקודם!) מצהיר **הצהרה חדשה**, כי לפנינו מסך גרפי 0, ואגב כך נמתחת יחידה אופקית ראשונה של גרפיקה 0.

מקטע ד2 ממשיך ומותח עוד 3 שורות גרפיקה (כדי להשלים את החלון הקטן בן 4 שורות הטקסט).

מקטעים ג1, ו-ג2 מורים ל-Antic היכן נמצאים, בהתאמה, זיכרון המסך עבור האזור הגרפי, וזיכרון המסך עבור אזור הטקסט שבחלון הקטן.

מקטע ה' מכיל כזכור את פקודת הדילוג (המקודדת: 65), אשר מורה ל-Antic לחזור אל כתובת הזיכרון הראשונה של ה-Display List ולהתחיל הכול מחדש.

■ **ועתה משימה – הפוך את היוצרות!**

כתוב תוכנה קצרה שתשנה את ה-Display List הקיים, כך שחלון הטקסט הצר יופיע בראש המסך, ומתחתיו יופיע אזור הגרפיקה.

[תשובה 2 בעמוד 105]

ודאי שמת לב כי המסך התנהג מעט מוזר לרגע קט לפני שהושלמה המשימה. כדי להימנע מהתופעה הזו, תוכל לכבות לרגע את ה-Antic לפני ביצוע המשימה, ולהדליקו בחזרה מיד עם סיומה.

דע לך כי הצבת הערך 0 בתוך בית 559 תכבה את ה-Antic (ותשחיר את המסך). הערך 34 ישיב את ה-Antic לפעולה.

לפני שנעבור לסעיף הבא, אסב את תשומת לבך לאזור מיוחד בזיכרון המסך ששמור לחלון הטקסט. בכל המסכים הגרפיים שבהם מופיע חלון טקסט שכזה, יוקצו עבורו **תמיד 160 הבתים האחרונים** של זיכרון המסך, ממש לפני ה-RAMTOP ותחילתו של זיכרון ה-ROM.

מכיוון שכך, כאשר יש במסך הגרפי חלון טקסט, ה-Display List של המסך יכלול תמיד את קוד ההכרזה 66 ומיד אחריו את הערכים 159 ,96 המורים בדיוק על המקום ה-160 לפני ה-RAMTOP.

מסכים לפי הזמנה

עד כה, כאשר פיתחת משחקים או תוכנות גרפיות בבייסיק הוגבלת לאחד מ-16 המסכים הקיימים.

עתה יכולות להיפתח לפניך אפשרויות רבות נוספות.

נניח שתרצה לבנות תמונת נוף כרקע למשחק. בחלקו העליון של המסך תוכל להסתפק בגרפיקה פשוטה יחסית לציור השמים למשל. בחלק המרכזי תרצה רמה מורכבת יותר של גרפיקה לציור מבנים, כבישים וכיוצא באלו, ובתחתית המסך תרצה אזור טקסטואלי להצגת ניקוד, פסילות וכדומה.

אם נחזור לאנלוגיה בה פתחנו את הפרק – אתה האדריכל וככזה תוכל לעצב את המסך כרצונך.

כמו אדריכל מקצוען המשרטט לעצמו תכנית מתאר, אציע לך לקחת דף נייר ולרשום עליו את מתווה התצוגה שלך – כיצד המסך שלך צריך להיראות, כלומר כמה יחידות אופקיות מכל סוג יופיעו בו.

בשלב התכנון קח בחשבון כי למסך שלך יש גובה מוגדר של 192 פיקסלים, כלומר 192 קווי רוחב ברוחב של פיקסל אחד, ואין לחרוג ממספר זה. כך למשל, אם כל שורת טקסט (של גרפיקה 0) היא בגובה של 8 פיקסלים, נחלק 192 ב-8, ונקבל 24 שורות.

כל עוד אתה מתכנת בסביבת העבודה של בייסיק, אין צורך "להמציא את הגלגל". תוכל להשתמש ב-Display List הקיים (כמו גם בהקצאת זיכרון המסך הנקבעת אוטומטית) ולהכניס ל-Display List את השינויים הנדרשים.

כלל האצבע שצריך להנחות אותך הוא לבחור ב-Display List של הגרפיקה העדינה ביותר שתעשה בה שימוש. לדוגמא, אם ברצונך לבנות

מסך שיש בו אזור גרפי מגרפיקה 3, אזור גרפי מגרפיקה 6 ואזור טקסט מגרפיקה 1, השתמש ב-Display List של גרפיקה 6 (הגרפיקה העדינה ביותר), והכנס לתוכו את השינויים הנדרשים.

להלן טבלה המסכמת את ערכי הקידוד של כל יחידה אופקית גרפית, ואת מספר השורות (ברוחב פיקסל אחד) המרכיבות אותה:

מצב גרפי (בבייסיק)	קוד "הכרזה"	קוד יחידה	גובהה (בפיקסלים) של כל יחידה	תאי זיכרון ליחידה
Gr. 0	66	2	8	40
Gr. 1	70	6	8	20
Gr. 2	71	7	16	20
Gr. 3	72	8	8	10
Gr. 4	73	9	4	10
Gr. 5	74	10	4	20
Gr. 6	75	11	2	20
Gr. 7	77	13	2	40
Gr. 8	79	15	1	40
Gr. 9	–	–	1	40
Gr. 10	–	–	1	40
Gr. 11	–	–	1	40
Gr. 12	68	4	8	40
Gr. 13	69	5	16	40
Gr. 14	76	12	1	20
Gr. 15	78	14	1	40

נתבונן למשל על השורה השביעית בטבלה המפרטת את נתוניה של יחידה אופקית מסוג: Gr. 6

קוד ההכרזה של מסך גרפי 6 הוא 75, וכזכור, אגב ההכרזה על סוג המסך, נמתחת גם יחדה אופקית ראשונה מן הסוג הזה.

קוד היחידה הגרפית של מסך גרפי 6 הוא 11, וערך זה יחזור שוב ושוב לאורך ה-Display List.

? כמה פעמים?

ובכן, לפי הטבלה, גובהה של כל יחידה גרפית הוא 2 ולכן כדי למלא את כל 192 השורות הפעילות של המסך, נוכל להכניס 96 יחידות כאלו. ואולם עלינו להביא בחשבון שני דברים: ראשית, כבר הכנסנו יחידה אופקית אחת בשלב ההכרזה ולכן נותרו רק 95 יחידות גרפיות למילוי המסך כולו.

ושנית, אם ברצוננו להשאיר חלון טקסט סטנדרטי בן 4 שורות בתחתית המסך יש להקצות עבורו 32 קווי פיקסל (שכן: 4X8=32) כמובן שקווי פיקסל אלו באים על חשבון היחידות הגרפיות, וחישוב פשוט מראה כי בסך הכול עלינו לגרוע 16 יחידות גרפיות מן המסך.

■ הצג עתה את ה-Display List של מסך גרפי 6, ובדוק אם אכן קוד ההכרזה שלו הוא 75, ואם קוד היחידה 11, חוזר בדיוק 79 פעמים!

אגב, בוודאי שמת לב כי בכל אחד מן המצבים הגרפיים קוד ההכרזה גדול תמיד מקוד היחידה ב-64. אם תרצה להבין מדוע, תוכל לעיין בנספח 1 להלן.

לאחר שהתנסינו בקריאת הטבלה, נוכל לבנות עתה מסך לפי הזמנה. נניח שברצוננו לבנות מסך המורכב משלוש יחידות: חלקו העליון יכלול 4 שורות טקסט מסוג: **Gr. 1**, מתחתן יהיו 8 יחידות גרפיות מסוג: **Gr. 3** ושאר המסך יכיל 48 יחידות גרפיות מורכבות יותר מסוג: **Gr. 6**

ראשית, נחשב ונוודא שהמשימה עומדת בתנאי הגובה המקסימליים:

(4x8) + (8x8) + (48x2) = 192

שנית, נרשום (רצוי על דף נייר) מתווה תצוגה מפורט הכולל את קודי ה-Display List הרלוונטיים. בשלב זה נשאיר מקום ריק עבור הקצאת האזורים בזיכרון המסך:

א	112	112	112				
ב1	70						
ג1							
ד1	6	6	6				
ב2	72						
ג2							
ד2	8	8	8	8	8	8	8
ב3	75						
ג3							
ד3	11	11	11 ... x 47				
ה	65	130	151				

זכור. כל מקטע פותח בקוד ההכרזה הרלוונטי, ובהמשך מופיע מספר היחידות הנדרש, פחות אחד!

לבסוף, נעבור לשלב מדידת כמות הזיכרון הדרושה לכל מקטע.

- 4 יחידות של גרפיקה 1 דורשות 80 תאי זיכרון (20x4)
- 8 יחידות של גרפיקה 3 דורשות 80 תאי זיכרון (10x8)
- 48 יחידות של גרפיקה 6 דורשות 960 תאי זיכרון (20x48)

בסך הכול נדרשים 1,120 תאים של זיכרון מסך.

מכיוון שאנחנו מתכנתים בסביבת עבודה של הבייסיק, נעדיף, כאמור, לתת למחשב להקצות עבורנו את כמות זיכרון המסך (גם אם ההקצאה הזו בזבזנית וכמה מאות תאי זיכרון "הולכים לאיבוד"). לפי כלל האצבע שהצגנו קודם, ניעזר במסך גרפי 6 (הגרפיקה העדינה ביותר שאנו עושים בה שימוש) ונעשה שינויים על ה-Display List שלו.

את הכתובת של תחילת זיכרון המסך נוכל למצוא בעזרת זוג הבתים המצביעים: 88, 89. כל שעלינו לעשות הוא לפקוד על המחשב: **Gr . 6** ולרשום (תחת הרובריקה: ג1) את הערכים של זוג הבתים המצביעים הללו.

אחר כך, נחשב את כתובות הזיכרון של שתי נקודת ההכרזה הבאות:

כתובת הזיכרון תחת הרובריקה ג2 תהייה 80 בתים מעל זוג הערכים שרשמנו תחת הרובריקה ג1, ואילו כתובת הזיכרון תחת הרובריקה ג3 תהייה 960 בתים נוספים מעל.

נמלא במתווה התצוגה שלנו את הנתונים החסרים:

א	112	112	112					
ב1	70							
ג1	*224*	*151*						→ *38880*
ד1	6	6	6					
ב2	72							
ג2								*(+80) = 38960 → 48,152*
ד2	8	8	8	8	8	8	8	
ב3	75							
ג3								*(+80) =39040 → 128,152*
ד3	11	11	11					... X 47
ה	65	130	151					

שים לב למהלך החשבוני שעשיתי: את כתובת הזיכרון הראשונה חישבתי בשיטת ה-MSD, LSD : 151X256+224=38880. על מנת לקבל את כתובת הזיכרון השנייה, הוספתי לתוצאה 80 בתים וקיבלתי 38960 שהם לפי שיטת ה-MSD, LSD : 162 ,48. וכן הלאה.

אחרי שרשמנו את כל הנתונים הדרושים למתווה התצוגה שלנו, הגיע הזמן להקלידם למקומם המתאים.

■ כתוב תוכנית קצרה אשר תייצר את המסך המורכב שלנו. אל תשכח לכבות לרגע את ה-Antic כדי להימנע מהתנהגות לא רצויה של המסך.

[תשובה 3 בעמוד 105]

אם פעלת כשורה, אתה אמור לראות בשלב זה מסך שחור לחלוטין. כדי לבדוק את מבנה המסך, נמלא אותו בכמה סימנים גרפיים.

שים לב, אם תרצה להשתמש בפקודות: **PLOT** ו-**DRAWTO** ואפילו **PRINT#6**, תצטרך לעזור ל-Antic "להתמצא במרחב". זאת משום שכאשר פקדת על המחשב: **GR.6** ה-Antic "יודע" שעליו להתנהג לפי החוקים של מסך גרפי 6, גם אם בפועל השתנו האיכויות הגרפיות של המסך!

לעזרתנו יכול לשמש הבית 87. בית זה מורה ל-Antic מהו המסך הגרפי הפעיל. אם נשנה את ערכו של בית 87 נוכל "לבלבל" את ה-Antic ולגרום לו להתנהג לפי רצוננו, בתוך המסך המורכב שיצרנו.

■ הוסף לתוכנה את שורות הקוד הבאות:

```
70 POKE 87,1:? #6;"FOUR":? #6;"THREE":? #6;"TWO":? #6;"ONE"
80 POKE 87,3:COLOR 2:PLOT 0,8:DRAWTO 20,15
90 POKE 87,6:COLOR 1:PLOT 80,8:DRAWTO 0,52
```

שים לב. שורה מספר 70 מורה ל-Antic להתייחס למסך **כולו** כאל גרפיקה 1, ומשעה שהוא עושה כן, נוכל לכתוב ארבע שורות טקסט בחלקו העליון.

באותו האופן, שורה מס' 80 מורה ל-Antic להתייחס למסך **כולו** כאל גרפיקה 3, ושורה מס' 90 מורה ל-Antic להתייחס למסך **כולו** כאל גרפיקה 6.

ועדיין השימוש בפקודות: **PLOT**, **DRAWTO** וכו' **לא מומלץ** במסכים המורכבים, ונעדיף את הכתיבה הישירה אל תוך זיכרון המסך. בפרקים הבאים אדגים זאת.

טיפול במסכים מעט יותר מסובכים

א. DISPLAY LIST ארוכים

עד כה חקרנו את מסכי הטקסט: 0, 1, 2, וגם כמה מסכים גרפיים פשוטים יחסית. אם תבדוק את ה-Display List של שמונה המסכים הראשונים בבייסיק (0-7), תראה שהם בנויים כולם לפי אותה שיטה שסקרתי קודם. במסכים הגרפיים המורכבים יותר – 8, 15 התמונה מעט משתנה.

■ פקוד על המחשב לרשום את ה-Display List של מסך 8.

שים לב, מכיוון שמדובר ברשימה ארוכה שלא תיכנס לחלון הצר בתחתית המסך, אציע לך להיעזר במערך מספרי ולשמור בתוכו את הערכים. הגדר את המערך שלך בגודל של 200 תאים.

[תשובה 4 בעמ' 106]

התבונן ברשימת הערכים שקיבלת. כצפוי הערך 15 (המקודד את היחידה הגרפית של גרפיקה 8) חוזר שוב ושוב 158 פעמים. אלא, שאחרי 93 חזרות שכאלו ה-Display List לוקח "פסק זמן", מכריז מחדש על קוד המסך (79) וממשיך ברצף של עוד 65 חזרות.

? מדוע זה קורה?

ובכן, ה-Antic שלנו הוא מעבד גרפי מוגבל ביכולותיו והוא אינו מסוגל לטפל בלמעלה מ-4096 תאי זיכרון מסך ברצף אחד (כלומר בלמעלה מ-4K של זיכרון מסך).[3]

מסך 8 הוא מסך גרפי עמוס מידע, המשתמש בכמעט 8K של זיכרון מסך ועל כן ה-Antic חייב בשלב מסוים לרענן את עצמו – למחוק את כל המידע השמור בתוכו, ולהיטען במידע חדש. או במילים אחרות, להשחיר את חציו העליון של מסך הטלוויזיה, ורק אז להתחיל לצייר את חציו התחתון.

אך אל דאגה! אינך צריך להיכנס למרה שחורה בגלל חצי התמונה השחורה... מכיוון שכל התהליך הזה של מעבר בין חלקו האחד של המסך לחלקו השני

3 בנוסף, קח בחשבון כי ה-Antic אינו מסוגל לטפל ב-Display List שאורכו גדול מ-1024 בתים.

מתרחש שוב ושוב במהירות הבזק (כזכור, קצב ריענון המסך הוא 50 פעמים בשנייה!) העין האנושית לא מבחינה בריצוד כלשהו ותמונת המסך נראית שלמה ורציפה.

התנהגות דומה תמצא גם בגרפיקה מספר 15.

כידוע לך מסכים גרפיים 14 ו-15 מאפשרים שימוש ברזולוציה הגבוהה ביותר האפשרית: 160X160, אך בעוד שמסך גרפי 14 מאפשר שימוש ב-2 צבעים בלבד, הרי שמסך גרפי 15 מאפשר שימוש ב-4 צבעים ולכן דורש זיכרון מסך גדול יותר – של כמעט 8K.

■ פקוד על המחשב להציג את ה-Display List של מסכים 14 ו-15 וראה את ההבדל ביניהם.

ב. משלבים כוחות בין ה-ANTIC וה-GTIA

ודאי יצא לך להתנסות גם בשלושת המסכים הגרפיים 9, 10, 11.

למסכים הללו ישנם שלושה מאפיינים ייחודיים:

(1) הם צבעוניים יותר מאחרים – מסך גרפי 9 מאפשר לך להשתמש ב-16 גוונים של אפור; מסך גרפי 11 מאפשר לך להשתמש ב-16 צבעים שונים; ואילו מסך גרפי 10 מאפשר לך להשתמש ב-9 גוונים שונים (הנשלטים על ידי ערכי הבתים 705-712).

(2) אין בהם חלון טקסט תחתון.

(3) הם "נעלמים" מיד עם עצירה/סיום של התוכנה.

■ פקוד על המחשב להציג את ה-Display List של כל אחד משלושת המסכים הללו.

התוצאה בשלושת המקרים זהה! והיא דומה להפליא ל-Display List של מסך גרפי 8! (למעט, כמובן, המקטע הפוקד על הוספת חלון הטקסט התחתי בן 4 שורות).

? היכן בכל זאת טמון ההבדל?

ובכן, בשלושת המצבים הגרפיים הייחודיים הללו, נכנס לפעולה בנוסף למעבד ה-Antic גם מעבד ה-GTIA ושילוב הכוחות בין המעבדים מאפשר להשתמש באותה הרזולוציה של מסך גרפי 8, אך בתוספת הצבעונית הייחודית.

דע לך כי הבית האחראי להפעיל את ה-GTIA באחד משלושת המצבים הצבעוניים הללו הוא 623 (שאגב, כבר פגשנו אותו בהקשר אחר בפרק ב של מחשבת 7). כאשר אנו עובדים במסך גרפי 9, בית 623 מקבל את הערך 64. כאשר אנו עובדים במסך גרפי 10, בית 623 מקבל את הערך 164. וכאשר אנו עובדים במסך גרפי 11, בית 623 מקבל את הערך 228.

תוכל להשתעשע ולפקוד את הערכים הללו – 64, 164 ו-228 לתוך בית 623 גם כשאתה עובד במסך גרפי 0.

■ בדוק כיצד מגיב המסך כאשר אתה משנה הגדרות אלו של את ה-GTIA.

כדי להשיב את מצב ה-GTIA לקדמותו, פקוד לתוכו את הערך 32 או פשוט הקש על מקש RESET

סיכום וחזרה

בפרק זה הכרנו את מתווה התצוגה (Display List) – תוכנית המתאר של המסכים הגרפיים השונים ולמדנו לקרוא ולהבין את הוראותיו:

- כל Display List פותח בשלוש יחידות שחורות (המקודדות בערך 112).
- לכל אחת מן היחידות הגרפיות יש "קוד הכרזה" משלה, ובנוסף קוד ייחודי שבו היא מתכנה (ואשר חוזר שוב ושוב ברצף).
- עם פתיחתו של כל מקטע גרפי מנותב ה-Antic לכתובת ייחודית בזיכרון המסך) בשיטת ה-MSD, LSD.
- כל Display List נחתם בפקודת דילוג (המקודדת בערך 65) ומיד לאחריה מנותב ה-Antic אל כתובת תחילת ה-Display List.

בהמשך הפרק סקרנו את ה-Display Lists המובנים בבייסיק, למדנו לערוך בהם שינויים ואף להפוך את המסכים הגרפיים שלנו למסכים מורכבים ומרובי מקטעים.

זכור כי כל בנייה של Display List מתחילה בדף נייר ועיפרון, והיא מחייבת חישוב מדוקדק של מספר היחידות הזמינות במסך, ושל גודל הזיכרון המוקצה לכל מקטע גרפי.

בפרק הבא נלמד לנצל את ה-Display List ליצירת אפקט גלילה (Scrolling).

פרק ב
גלילה (SCROLLING)

בפרק הקודם הכנסתי אותך לנעליו של "אדריכל". עתה אציע לך להחליף מקצוע ולדמות בנפשך כי אתה "במאי קולנוע"!

ובכן, נניח כי אתה מבקש להפיק סרט פעולה הפותח בסצנה של נסיעה מהירה בכביש, אלא שהתקציב העומד לרשותך דל במיוחד ואין באפשרותך להוציא את צוות הצילום שלך מן האולפן.

אחת האפשרויות שעומדות לרשותך היא להשאיר את המכונית במנוחה, בתוך האולפן הממוזג, ולצלם אותה כשמאחוריה מוקרן כל העת נוף בתנועה. תזוזה של הנוף **לאחור** תיצור אצל הצופה אשליה שהמכונית נעה קדימה, ואילו תזוזה של הנוף **קדימה** תיצור אצל הצופה אשליה שהמכונית נוסעת לאחור.

אפקט דומה נוכל ליישם גם על מסך הטלוויזיה שלך. על מנת ליצור תנועת גלילה על המסך נוכל להעתיק – קדימה או אחורה – את מקומם של כל התאים בזיכרון המסך בעת ובעונה אחת, ולעשות זאת שוב ושוב.

כמובן שפעולת העתקה שכזו תהיה איטית להפליא (זכור שמדובר בכמה מאות ואפילו אלפי תאי זיכרון) ולכן נפעל בדרך שונה. במקום להעתיק את כל התאים של זיכרון המסך, נוכל להותיר את זיכרון המסך על כנו ולהזיז קדימה או אחורה אך ורק את **כתובת הזיכרון** המצביעה עליו.

כפי שראינו בפרק א', ה-Display List פותח תמיד בחזרה שלוש פעמים על הערך 112. אחר כך, במקום הרביעי ב-Display List מופיעה ההכרזה על סוג

המסך הגרפי ואילו המקום החמישי והשישי ברשימה מאכסנים את כתובת הבית הראשון של זיכרון המסך אליה מופנה ה-Antic.

אלא שאם נמסור ל-Antic כתובת שונה, הוא יארגן מחדש – **כהרף עין** – את המסך לפי הכתובת החדשה.

■ שנה את כתובת הזיכרון ב-Display List והעתק אותה קדימה ביחידה אחת.

[תשובה 5 בעמ' 106]

שים לב: כאשר הזזת קדימה את כתובת הזיכרון, כל הסימנים הגרפיים על המסך הוסטו בעת ובעונה אחת אחורה – בדיוק כמו באפקט הנוף המתנועע מאחורי הרכב באולפן הצילומים.

הזזת כתובת הזיכרון יכולה כמובן להתבצע שוב ושוב עד שכל הסימנים הגרפיים בצדו האחד של המסך ינועו מצדו האחד של המסך אל צדו השני.

■ כתוב עתה תוכנה אשר "תסיע" שורת טקסט קצרה לרוחב המסך. על מנת ליצור את אפקט "שלט החוצות" הזה, מקם את הטקסט שלך בשורה השנייה והמשך להסיע אותו עד שיעלם בצדו השמאלי של המסך ויחזור מצדו הימני.

[תשובה 6 בעמ' 107]

ודאי שמת לב שתנועת הטקסט על המסך אינה משביעת רצון. ראשית, הטקסט אינו "נבלע" ונעלם בצדו השמאלי של המסך (כפי שהיינו מצפים שיקרה בתנועת גלילה), ואם לא די בכך, כשהטקסט שב ומופיע בצד השמאלי של המסך, הוא לא שומר על גובהו אלא "מטפס" שורה אחת למעלה.

על מנת לפתור את התקלה עלינו להרחיב את היריעה...

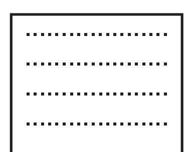
א. מרחיבים את היריעה

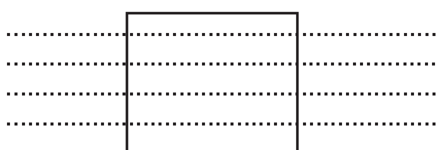
ככלל, אם ברצוננו לגלול דבר מה (טקסט, ציור, תמונה), האובייקט הנגלל חייב להיות ארוך יותר מאזור התצוגה. חשוב למשל על ספר התורה הכתוב על קלף. החלק הנגלל הוא רצועה ארוכה מאוד, אך רק חלק קטן ממנה נפרש לעיניו של הקורא, בעת הקריאה.

כפי שראית בפרק א', המסכים הגרפיים בנויים בתכנון מוקפד וחסכני. כשמסתיימת יחידה אופקית אחת, מתחילה מיד היחידה הבאה בתור. זאת ועוד זאת, אורכן של היחידות האופקיות קבוע ולא ניתן לשינוי (בטבלה המופיעה בפרק א' מצוין מספר תאי הזיכרון שמכילה כל אחת מן היחידות הגרפיות השונות).

נתבונן למשל על מסך גרפי 0. כל יחידה אופקית מכילה 40 תאי זיכרון ולכן תא הזיכרון ה-41 כבר שייך לשורה השנייה. ואולם, על מנת ליצור אפקט גלילה על המסך היינו רוצים מספר גבוה יותר של תאי זיכרון – אשר יחרגו לשני הצדדים, מעבר לאזור התצוגה.

נדגים זאת באמצעות שני איורים: האיור הימני מציג את המצב הקיים, ואילו האיור השמאלי מציג את המצב הרצוי לנו.





? כיצד אם כן נוכל לשנות דבר מה שלא ניתן לשינוי?

ובכן התשובה היא שלא נוכל! אבל כן נוכל להתערב ולקבוע את כתובת הזיכרון של כל יחידה ויחידה.

אסביר.

הערך הרביעי ב-Display List (לאחר מתיחת שלושת היחידות השחורות), הוא כזכור קוד ההכרזה על היחידה הגרפית, ומיד אחריו מצוינת כתובת תחילת זיכרון המסך. ואולם, אין שום מניעה שנחזור פעם נוספת או אפילו פעמים נוספות על קוד ההכרזה הזה, ובלבד שכל פעם נפנה את ה-Antic לכתובת זיכרון רלוונטית.

נתבונן למשל על מקטע מתחילת ה-Display List של מסך גרפי 0:

112, 112, 112, <u>66, 64, 156</u>, 2, 2, 2,...

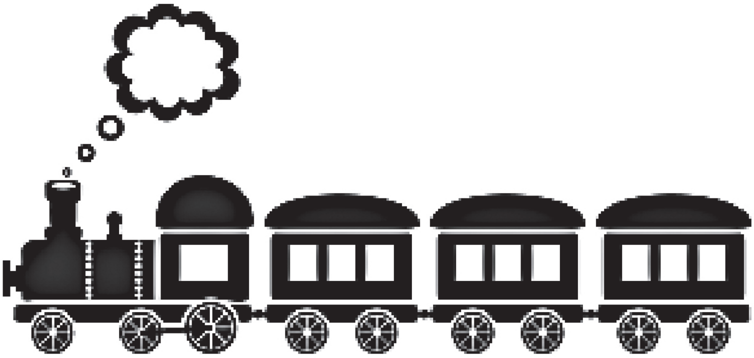
במקום לחזור בפשטות על קוד היחידה הגרפית (קרי על הערך 2), נוכל לחזור שוב ושוב על קוד ההכרזה (קרי: 66), ומיד אחריו לציין את כתובת הזיכרון הרלוונטית, כך:

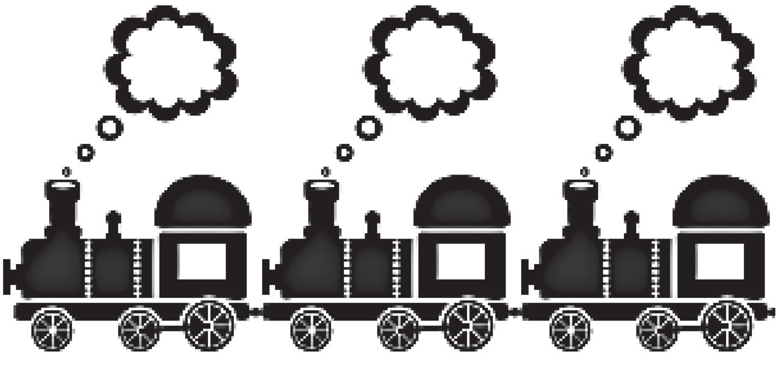
112, 112, 112, <u>66, 64, 156</u>, <u>66, 104, 156</u>, <u>66, 144, 156</u>, ...

שים לב. ה-Display List החדש נותן תוצאה זהה לחלוטין, אם כי הוא כמובן הרבה פחות "חסכוני". ואולם, בשיטה הזו יש לנו הרבה יותר "יד חופשית". כך, איננו מחויבים לקדם את כתובת הזיכרון בכל שלב ב-40 תאים, אלא נוכל להתקדם בקפיצות גדולות יותר.

במובן זה, את ה-Display List הרגיל אפשר לדמות לרכבת סטנדרטית – החלק הראשון הוא הקטר המכריז על סוג הרכבת ועל היעד שלה (קרי: כתובת הזיכרון), ואילו שאר החלקים הם קרונות הנגררים אחריה בזה אחר זה.

לעומת זאת את ה-Display List החדש שיצרנו אפשר לדמות לרכבת מוזרה במקצת, הבנויה כולה מקטרים...





בדוגמא הבאה למשל, אנו קופצים ב-80 תאי זיכרון בכל שלב, וכך מרחיבים את היריעה פי שניים!

112, 112, 112, 66, 64, 156, 66, 144, 156, 66, 224, 156, 66, 48, 157, ...

שים לב, כאשר קפצנו בפעם השלישית ב-80 יחידות זיכרון, השתנה גם הבית המצביע השני של כתובת הזיכרון. שכן: 304=80+224. אך 304 גבוה מ-256 ב-48 יחידות).

נשוב עתה אל משימת "שלט החוצות".

מכיוון שהמשימה העומדת על הפרק היא "הסעה" של שורה אחת בלבד, אין צורך לחזור 24 פעמים על קוד היחידה הגרפית (66) אלא נוכל להסתפק בשלוש פעמים בלבד: פעם ראשונה, בתחילת ה-Display List (ממש אחרי שלוש החזרות של קוד היחידה הריקה 112); פעם שנייה כדי לקבוע היכן – בטווח זיכרון המסך – פותח המקטע המורחב שעליו נושיב את הטקסט המתנועע; ופעם שלישית כדי לציין היכן – בטווח זיכרון המסך – מסתיים אותו מקטע מורחב, וממנו ואילך ממשיכות יחידות הגרפיקה הרגילות (המקודדות בערך 2).

■ כתוב כעת תוכנה משופרת של "שלט חוצות". אל תשכח להתחיל, כמו אדריכל מיומן, בתכנון עם דף ועיפרון!

אך לפני שאשחרר אותך לדרכך, אתן לך שני רמזים:

ראשית, עליך לקחת בחשבון את אורכו המוגדר של הזיכרון המוקצה ל-Display List של מסך מספר 0. בשלב זה של הלימוד, עליך לדאוג כי השינויים שתכניס ב-Display List לא יגדילו את אורכו (אחרת תהייה התנגשות לא רצויה עם אזור אחר בזיכרון). הפתרון הפשוט שאציע לך הוא פשוט להשמיט כמה שורות מהמסך ולקצר אותו מעט.

שנית, זכור כי מסך גרפי 0 מותיר שוליים של שני תווים בצדו השמאלי של המסך, והפקודה **PRINT** מתחשבת בשוליים הללו כשהיא מסדרת את

הטקסט על המסך. על מנת לאפס את השוליים הללו וליצור גלישת טקסט רציפה, פקוד על המחשב:

```
POKE 82,0
```

[תשובה 7 בעמ' 107]

ב. מסך שלם רחב-יריעה

עתה, כשהבנו את עיקרון הגלילה של שורת טקסט אחת, נסבך את העניינים וננסה לגלול את **כל** הגרפיקה המצוירת על המסך.

לצורך המשימה, נצייר מעגל גדול על אחד המסכים הגרפיים ובהמשך ננסה לגלגל אותו לרוחב המסך.

■ צייר על מסך גרפי 3 מעגל גדול שנוגע בקצהו הימני של המסך, וקוטרו כגובה המסך. אם אינך זוכר כיצד עושים זאת, רענן את זיכרונך בפרק ג' של מחשבת 4: "גרפיקה מתמטית".

[תשובה 8 בעמ' 107]

בשלב זה אתה ודאי יודע שעל מנת לגלול כל אחת מן היחידות הגרפיות, יש להגדיר עבורה מקטע מורחב בזיכרון המסך ואגב כך גם להכריז בכל פעם מחדש את קוד ההכרזה 72 של מסך גרפי 3.

על מנת לגלול את המסך כולו, נדרשות אפוא 20 "נקודות עוגן" שכאלו – כמספר היחידות הגרפיות שמהן בנוי המסך. כמובן שיצירת כל כך הרבה נקודות עוגן מנפחת את גודלו של ה-Display List. חישוב מהיר מראה כי גודלו

של Display List מנופח שכזה הוא 72 תאים (וזאת בהשוואה ל-34 תאים של מסך גרפי 3 הרגיל).

בנוסף, על מנת ליצור את אפקט הגלילה נצטרך כמובן להרחיב את היריעה. כך למשל, הגדלת אורכה של כל יחידה גרפית מ-10 תאים ל-40 תאים, דורשת כמות זיכרון מסך הגדולה פי 4 מכמות הזיכרון הזמינה במסך גרפי 3 הרגיל!

כדי לפתור את שתי הבעיות הללו, נוכל "לצאת לדרך" ממסך גרפי "כבד" יותר. מסך גרפי 6 למשל עשוי להיות בחירה טובה, שכן המחשב יודע להקצות עבורו כמות זיכרון מסך גדולה יחסית, וגם אורך ה-Display List המוקצה לו גדול דיו לצורך המשימה וזאת מבלי שנסתכן ב"התנגשות" עם מקטעי זיכרון אחרים.

אך בכך לא תמו בעיותינו... זכור כי לכל אחת מ"נקודות העוגן" שתיצור מקושרת כתובת של זוג בתים מצביעים, וקח בחשבון כי על מנת לקדם את אותה "נקודת עוגן", נדרש חישוב מתמטי לשם קביעת ערכם החדש של זוג בתים אלו. כל עוד מדובר בהזזה של יחידה גרפית אחת, פעולת החישוב הזו אינה גוזלת הרבה זמן, אך כשמדובר בהזזה של 20 יחידות במקביל, סביר להניח שבמקום גלילה חלקה, ייראה על המסך מעין "ריקוד" איטי של הגרפיקה.

בשלב זה, יפה השעה לעזוב לרגע את סביבת הבייסיק ולנצל את הידע שרכשנו ביחידת הלימוד 8 "צעדים ראשונים בתכנות באסמבלי".

■ ראשית כל, פקוד על המחשב לעבור למסך גרפי 6, ורשום לפניך את הכתובת בה מתחיל זיכרון המסך, ואת הכתובת בה יושב ה-Display List:

כתובת ה-Display List: 256*151+130=38786

כתובת תחילת זיכרון המסך: 256*151+224=38880

■ שמור את התוכנה המציירת מעגל על מסך גרפי 3. עוד נשוב אליה בהמשך. עתה, טען את הקרטריג' Assembler Editor.rom; אל כונן הדיסקטים טען דיסקט עם מערכת ההפעלה DOS; וודא כי אתה עובד במצב בייסיק.

לפני שנצלול לעומק התכנות באסמבלי אדגיש כי איננו נפרדים לשלום מן הבייסיק. סביבת העבודה שלנו נשארת סביבת הבייסיק, לטוב ולרע, ואילו האסמבלי ישמש אותנו ככלי עזר. כפועל יוצא מכך, נקפיד להוסיף לשורות הקוד שנכתוב באסמבלי את הפקודות הנדרשות להפעלתן מתוך הבייסיק, וזאת על מנת שנוכל להריץ את הרוטינות באמצעות פקודת הבייסיק: **USR**.

■ נסה לכתוב בכוחות עצמך רוטינה שתפעל מתוך מסך גרפי 6, ותהפוך אותו למסך גרפי "רחב יריעה" הבנוי מיחידות של גרפיקה מס' 3. היעזר בנתונים שרשמת לפניך על אודות כתובת זיכרון המסך וכתובת ה-Display List. אל תשכח להוסיף את שתי הפקודות (הראשונה והאחרונה) אשר הופכות את התוכנה לרוטינה הנקראת מן הבייסיק.

בכל מקרה, עיין גם בהסבר המפורט שלי...

בשלב הראשון, נגדיר תווית ונשמור בה את כתובת ה-Display List, ובנוסף זוג תוויות בהן נשמור את הכתובת בה מתחיל זיכרון המסך (הכול, כמובן בשיטה ההקסה-דצימלית).

```
10    *=$600
20 DLST=$9782
30 MSD=$97
40 LSD=$E0
```

על מנת לייעל את תהליך הכנסת נתוני ה-Display List למקומם, נתקדם צעד אחר צעד באמצעות פקודת המנייה: **INX**. אך במקום לחזור שוב ושוב על פקודת המנייה ועל הכנסת הנתון למקומו, ניעזר בסאב-רוטינה, ונכנה אותה בתווית: **DO**)

```
360 DO
370     STA DLST,X
380     INX
390     RTS
```

שלושת הערכים הראשונים שעלינו להכניס הם כזכור: 112 (או $70). נאפס אפוא את המונה ונכניס אותם למקומם, תוך הפנייה לסאב-רוטינה:

```
60      LDX #$0
70      LDA #$70
80      JSR DO
90      JSR DO
100      JSR DO
```

הערך הבא שיש להכניס הוא קוד ההכרזה על גרפיקה 3, הלוא הוא: 72 (או $48). ומיד לאחריו שני הבתים המצביעים על כתובת זיכרון המסך הראשונה.

שים לב, מכיוון שאנו שבים ומקפיצים בכל שלב את כתובת הזיכרון קדימה, עלינו לחשב בכל פעם מחדש את ערכם של זוג הבתים המצביעים. מכיוון שכך, אנו חייבים להניח לרגע בצד את הערך השמור בכל אחד מהבתים המצביעים וליטול אותו בהמשך. לצורך כך תשמש אותנו המחסנית:

```
110     LDA  #$48
120     JSR  DO
130     LDA  #LSD
140     PHA
150     JSR  DO
160     LDA  #MSD
170     PHA
180     JSR  DO
```

עד כה התקדמנו שישה צעדים בכתיבת ה-Display List שלנו, ונותר לנו לכתוב את 19 "נקודות העוגן" הנוספות, כל אחת ממוקמת 40 צעדים קדימה מקודמתה.

לצורך כך, נבנה לולאה מותנית ובתוכה נחשב את ערכי זיכרון המסך הרלוונטיים ונציב אותם במקום. 19 "נקודות עוגן" שכאלו תופסות 57 מקומות ב-Display List, ואם נחבר את 6 הצעדים שכבר עשינו, נקבל 63 מקומות. כאשר נגיע למקום ה-63 נדאג לדלג החוצה מן הלולאה.

שים לב למהלך בשורות 220-320: אנו שולפים את שני הנתונים ששמרנו מבעוד מועד במחסנית, את הבית הגבוה (MSD) אנו שומרים ברגיסטר Y, ואילו את הבית הנמוך (LSD) אנו שומרים באוגר. כל שנותר לנו לעשות עתה הוא להוסיף לערכו של הבית הנמוך 40 יחידות, ואת הבית הגבוה להגדיל ב-0 יחידות, תוך הסתמכות על מצב דגל השארית C (כפי שלמדנו לעשות בפרק ג במחשבת 8). כמובן שמיד עם הצבת הנתון המחושב למקומו ב-Display List, חייבים לאחסן אותו מחדש במחסנית, עבור השלב הבא, וחוזר חלילה.

```
190 LOOP
200      LDA  #$48
210      JSR  DO
220      PLA
230      TAY
240      PLA
250      CLC
260      ADC  #$28
270      PHA
280      JSR  DO
290      TYA
300      ADC  #$0
310      PHA
320      JSR  DO
330      CPX  #$3F
340      BEQ  CLOSE
350      JMP  LOOP
```

בשלב זה עלינו להכניס ל-Display List את פקודת הקפיצה, המקודדת כזכור בערך 65 (או $41), ולהודיע ל-Antic להיכן עליו לחזור כדי לקרוא מחדש את הוראות ה-Display List.

שים לב לפעולות החשבון אשר ניתן לבצע **בכתבן האסמבלר** על התוויות: בשורה 470 אנו מורים למחשב לקחת את ערך התווית ולחלק אותו ב-256, ואילו בשורה 440 אנו מורים למחשב לקחת את ערך התווית, לחלק אותו ב-256, ולשמור רק את השארית שנותרת אחרי החלוקה – לצורך כך משמש האופרטור: **&**.

```
400 CLOSE
410     LDA #$41
420     STA DLST,X
430     INX
440     LDA #DLST&255
450     STA DLST,X
460     INX
470     LDA #DLST/256
480     STA DLST,X
```

לבסוף, נזכור לפקוד את שתי הפקודות שיהפכו את התוכנה שכתבנו לרוטינה הניתנת להרצה מתוך הבייסיק:

```
50      PLA
490     RTS
```

■ הפוך את קוד האסמבלי לרוטינה בשפת מכונה, שניתנת להרצה מתוך הבייסיק ושמור את הקוד של **שפת המכונה** באמצעות הפקודה **SAVE**. אחר כך, הוצא את הקרטריג', חזור לסביבת בייסיק ורשום לפניך את ערכי הרוטינה (כמובן בשיטה העשרונית)

אם אינך יודע כיצד עושים זאת, תוכל לעיין בנספח 2 להלן.

עתה נטען שוב את תוכנת הבייסיק היוצרת את המעגל, נוסיף לה פקודה המעבירה את המחשב למצב גרפי 6, ונטען אל תוך "דף 6" את ערכי הרוטינה הללו.

```
100 GRAPHICS 6
110 READ D
120 IF D=-1 THEN 260
130 POKE 1536+M,D
140 M=M+1
150 GOTO 110
260 U=USR(1536)
800 DATA 104,162,0,169,112,32,60,6,
32,60,6,32,60,6,169,72,32,60,6,169,
224,72,32,60,6,169,151,72,32,60,6
810 DATA 169,72,32,60,6,104,168,104,
24,105,40,72,32,60,6,152,105,0,72,
32,60,6,224,63,240,8,76,31,6,157,130
820 DATA 151,232,96,169,65,157,130,
151,232,169,130,157,130,151,232,169,
151,157,130,151,96,-1
```

אם תריץ כעת את תוכנית הבייסיק, כמובן שלא תראה דבר על המסך. ברגע שהמחשב פוגש (בשורת הבייסיק 60) פקודה המורה לו לפתוח מסך גרפי חדש, המעגל אשר יצרת בשורות 30-40 יימחק.

? כיצד נוכל להחזיר את המעגל שציירנו לתוך המסך החדש שיצרנו?

כפי שהסברתי בפרק א', השימוש בפקודות **PLOT**, **DRAWTO**, **POSITION** וכו' מסתבך כשעובדים במסכים מורכבים, והופך בלתי אפשרי לחלוטין כשמדובר במסכים בעלי ריבוי של נקודות עוגן ויחידות בפריסה רחבה.

ואולם תמיד נוכל לכתוב ישירות אל תוך זיכרון המסך!

יצירת גרפיקה על מסך בפריסה רחבה אפשרית אם כך בתהליך בן שני שלבים: בשלב הראשון יוצרים את האיור על המסך הגרפי הפשוט (בדוגמא שלנו, המעגל שעל מסך גרפי 3). פוקדים על המחשב לקרוא את כל המידע השמור בזיכרון המסך ושומרים את המידע הזה בתוך תא מחרוזתי גדול דיו. בשלב השני – לאחר שסיימנו לעצב את המסך בפריסה הרחבה – שולפים את המידע הגרפי ששמרנו ו"שופכים" אותו אל תוך זיכרון המסך החדש (כמובן, במיקום המתאים).

נוסיף, אם כן, את שורות הקוד הנדרשות עבור שלב "קריאת המידע" ועבור שלב "שפיכת המידע":

```
10 DIM GR(10*20)
60 SCR=PEEK(89)*256+PEEK(88)
70 FOR C=1 TO 10*20:GR(C)=PEEK(SCR+C):
NEXT C
520 SCR=PEEK(89)*256+PEEK(88): C=0
525 FOR O=0 TO (40*20)-40 STEP 40
530 FOR R=2 TO 11:C=C+1
540 POKE SCR+O+R,GR(C)
550 NEXT R:NEXT O
```

■ הרץ כעת את התוכנית.

המעגל אשר יצרת במסך גרפי 3 המקורי, מועתק אל המסך החדש שיצרת.

מה שנותר לנו עשות עתה הוא להזיז את כל עשרים "נקודות העוגן" מקום אחד קדימה שוב ושוב, על מנת ליצור גלילה של כל המסך.

לשם כך, נחזור שוב אל סביבת האסמבלר.

■ שמור את תוכנת הבייסיק על הדיסקט וחזור לעבוד באסמבלר. נסה לכתוב בעצמך את שורות הקוד אשר יקדמו את כל נקודות העוגן ביחידה אחת.

שים לב – את הרוטינה החדשה יש לכתוב כמובן במקום אחר בזיכרון (אך מומלץ עדיין בטווח של דף מספר 6), כך שלא תידרס הרוטינה הקודמת!

כרגיל, אציע לך לעיין גם בהסבר שלי.

בשלב הראשון נגדיר בתווית את מיקומו בזיכרון של ה-Display List. גם הפעם ניעזר ברגיסטר X כמונה: נאפס אותו, ונקדם אותו בשני צעדים, עד המקום השלישי ב-Display List.

```
10     *=$65A
20  DLST=$9782
40     LDX  #$0
50     INX
60     INX
```

עתה נבנה לולאה עבור הזזת עשרים "נקודות העוגן". הנתון הראשון של כל "נקודת עוגן" שכזו הוא כזכור קוד ההכרזה של המסך והוא לא משתנה. שני הנתונים העוקבים בתור נושאים את הכתובת הרלוונטית בזיכרון המסך שלנו, ונקדם אותם בשיטה שכבר למדנו.

```
70  LOOP
80     INX
90     INX
100     LDA  DLST,X
110     CLC
120     ADC  #$1
130     STA  DLST,X
```

```
140     INX
150     LDA  DLST,X
160     ADC  #$0
170     STA  DLST,X
210     JMP  LOOP
```

שים לב, בשורות 100 ו-150 אנו משתמשים בפקודה **LDA** בצורה שלא הוצגה ביחידת הלימוד 8. דע לך כי אין מניעה להציב באוגר ערך משתנה, כלומר, בכל שלב לטעון לתוכו את ערכו של תא הזיכרון העוקב. ואולם, יש לקחת בחשבון כי שימוש בפקודה **LDA** בצורה הזו משפיע על דגל השוויון Z!

? ומדוע זה אמור לעניין אותנו?

ובכן, בבואנו לכתוב את פקודת הדילוג מן הלולאה – גם הפעם אחרי 63 צעדים – עלינו לקחת בחשבון שלא נוכל להיעזר בדגל Z, כלומר לא נוכל להשתמש בפקודות **BEQ** או **BNE**.

ואולם, אין מניעה להשתמש באחת מפקודות הדילוג האחרות – **BCS** ו-**BCC**, אשר מבוססות על השימוש בדגל השארית C. נאפס אם כך את הדגל C ונבצע את הדילוג באמצעות הפקודה **BCS**:

```
180     CLC
190     CPX  #$3E
200     BCS  END
220  END
```

לפני שנבצע את הפקודה **ASM** נזכור להוסיף את הפקודות המבטיחות שהרוטינה תפעל מתוך תוכנת הבייסיק שלנו:

```
30    PLA
230    RTS
```

■ שמור את הקוד של **שפת המכונה** באמצעות הפקודה **SAVE**; אחר כך, הוצא את הקרטריג', קרא מן הדיסקט את הקוד ורשום אותו לפניך.

כעת נוסיף/נדרוס את השורות הקוראות את הרוטינה השנייה.

```
120 IF D=-1 THEN 160
160 M=0: RESTORE 900
170 READ D
180 IF D=-1 THEN 260
190 POKE 1626+M,D
200 M=M+1
210 GOTO 170
600 FOR LOOP=1 TO 20
610 U=USR(1626)
620 FOR TIME=1 TO 600: NEXT TIME
630 NEXT LOOP
900  DATA  104, 162, 0, 232, 232, 232, 232,
189, 130, 151, 24, 105, 1, 157, 130, 151, 232,
189, 130, 151, 105, 0, 157, 130, 151, 24
910 DATA 224, 62, 176, 3, 76, 95, 6, 96, -1
```

■ הרץ עתה את התוכנית ובחן את אפקט הגלילה שיצרת. מה דעתך?

ודאי שמת לב כי התנועה של המעגל היא תנועה בדילוגים והיא אינה רציפה וחלקה כמצופה מתנועת גלילה.

בסעיף הבא נעסוק בעניין זה בדיוק. בינתיים שמור את התוכנית שכתבת. נחזור אליה בהמשך...

ג. גלילה עדינה (FINE SCROLLING)

■ אתחל את המחשב מחדש במצב בייסיק.

המסך הפעיל כרגע הוא מסך מספר 0 וכפי שאתה יודע ה-Display List שלו כולל 23 יחידות הנושאות את הערך 2.

פקוד עתה על המחשב להחליף בחמישה מקומות בתוך ה-Display List את הערך 2, בערך 18:

```
POKE 39978,18
POKE 39979,18
POKE 39980,18
POKE 39981,18
POKE 39982,18
```

על פניו, תצוגת המסך לא השתנתה באופן מהותי (למעט סטייה קטנה בסידור השורות). אך דע לך שהשינוי שהכנסנו ב-Display List פתח למעשה 5 שורות לאפשרות של גלילה עדינה!

■ עתה הקלד כמה סימנים על המסך כדי שתוכל להבחין בנעשה עליו, ואחר כך פקוד על המחשב:

```
POKE 54276,1
POKE 54276,2
```

שים לב למה שמתרחש על המסך: האותיות בקצה השמאלי של המסך נבלעו פנימה ורק חלק מהם מבצבץ החוצה – יותר או פחות – בהתאם לערך שפקדת אל תוך בית 54276.

דע לך כי הבית 54276 אחראי על הגלילה העדינה האופקית. כאשר תפקוד לתוכו ערכים בסדר עולה, שורות **מסוימות** על המסך ינועו אט אט ימינה, פיקסל אחר פיקסל.

? אילו שורות בדיוק ינועו?

אותן שורות שהגדרנו ב-Display List כניתנות לגלילה עדינה.

? וכיצד נגדיר את השורות הניתנות לגלילה?

ובכן, כל שנדרש הוא להוסיף 16 לערך קוד ההכרזה על המסך הגרפי או לערך קוד היחידה הגרפית (או בשפה "מחשבית" יותר: "להדליק את הסיבית החמישית שלהם"[4]).

כך למשל, אם ברצוננו לגלול גלילה אופקית עדינה את השורה הרביעית של מסך גרפי 2, נקטע את רצף החזרות 7,7,7 ובמופע השלישי נכתוב את הערך 23 (שהוא תוצאת החיבור: 7+16).

באופן דומה, אם ברצוננו לגלול גלילה אופקית עדינה את השורה הראשונה של מסך גרפי 3, נשנה את קוד ההכרזה 72, ונכתוב במקומו את הערך 88 (שהוא תוצאת החיבור 72+16).

שים לב. הבית 54276 נמצא אי שם בתוך אזור ה-ROM של הזיכרון, וככזה הוא לא ניתן לשינוי על ידי המשתמש. יחד עם זאת ניתן לפקד לתוכו לרגע ערך מסוים. אם אינך מבין כיצד זה ייתכן, עיין בפרק א' ביחידת הלימוד "מחשבת 7".

---

4 דע לך כי קיימת שיטה נוספת לכנות את שמונה הסיביות של בית. בספרים מסוימים המתארים את תהליך הגלילה העדינה נהוג לכנות את הסיבית החמישית – D4, את הסיבית השישית – D5 ואת הסיבית השמינית – D7.

הגיע הזמן לשוב אל תוכנית גלילת המעגל שלנו ולשלב אל תוך הגלילה שבנינו, גם את הגלילה העדינה. כלומר, בכל אחד משלבי הגלילה, להוסיף סדרה של תזוזות קטנות של הגרפיקה.

? כמה תזוזות?

ובכן, הגלילה העדינה מאפשרת לנו בדיוק 15

■ הוסף כעת לתוכנת הבייסיק 15 שלבים של גלילה עדינה המשתלבים בגלילה האופקית הקיימת.

אל תשכח לעדכן את כל המופעים של קוד ההכרזה על המסך הגרפי ב-Display List ולהדליק בהם את הסיבית החמישית (קרי: להגדילם ב-16) לשם כך, עליך לחזור לכתבן האסמבלר, לעדכן את הרוטינה שכתבת ולהפוך אותה מחדש לקוד בשפת מכונה... למזלך, עשיתי זאת בשבילך:

```
612 FOR FINE=14 TO 0 STEP -1
615 POKE 54276,FINE
618 FOR TIME=1 TO 100: NEXT TIME
620 NEXT FINE
800 DATA 104,162,0,169,112,32,60,6,
32,60,6,32,60,6,169,88,32,60,6,169,
224,72,32,60,6,169,151,72,32,60,6
810 DATA 169,88,32,60,6,104,168,104,
24,105,40,72,32,60,6,152,105,0,72,
32,60,6,224,63,240,8,76,31,6,157,130
```

? מה דעתך?

הגלילה נראית סוף-סוף כמו גלילה של ממש, במובן זה שהיא רציפה ומהירה. ואולם, ודאי שמת לב שמשהו עדיין אינו כשורה – המסך מתנהג באופן קצת מוזר והוא מרצד...

על מנת להבין מדוע הריצוד הזה קורה, ניזכר בדימוי "קבלן הבניין" שהצגתי בפרק א'. כזכור, המיקרו-מעבד הגרפי – בדומה לקבלן הבניין – עסוק כל העת בציור המסך שורה אחר שורה. כשהוא מסיים את מלאכתו, הוא קופץ לנקודת ההתחלה ומתחיל הכול מחדש, שוב ושוב, חמישים פעמים בשנייה.

לקבלן החרוץ שלנו אין בעיה להתרוצץ שוב ושוב על פני המסך ולבצע את מלאכתו גם אם מחליפים לו מדי פעם את "תוכנית המתאר" ומוסרים לו מתווה תצוגה חלופי (כפי שאכן עשינו עשרים פעם!).

ואולם, הקבלן שלנו שונא שמפריעים לו באמצע העבודה! (ובינינו, מי אוהב שמפריעים לו באמצע?). וכך, ברגע שמזיזים את אבני הבניין שלו באמצע עבודת הבנייה, לפני שהספיק לסדר את כולן במקומן, הוא משתולל וכועס...

כדי לא להרגיז את קבלן הבניין החרוץ שלנו, עלינו פשוט להיות מנומסים יותר. להמתין בשקט. לתת לו לסיים את סבב הבנייה ולשוב בשלום אל נקודת ההתחלה, ורק אז, רגע לפני שהוא מתחיל מחדש בעבודתו הסיזיפית נוכל להזיז לו מעט את אבני הבניין...

ובמונחי המחשב – עלינו למצוא שיטה לבצע את המניפולציה של הגלילה האופקית העדינה, בסנכרון עם קצב ריענון המסך, כך שהגלילה העדינה תתרחש אך ורק בחלקיק השנייה שבו ה-Antic נח וממתין לסבב נוסף של בניית המסך.

? האם יש שיטה כזאת שבה ניתן "להפריע" ל-Antic אך ורק בנקודת זמן ספציפית?

התשובה היא כן, אך לא בבייסיק! גם הפעם עלינו להודות ולומר כי ככל שאנו נכנסים אל נבכי המחשב, כלי העבודה שלנו – שפת הבייסיק – הופך להיות פחות ופחות יעיל.

למזלנו, הידע שצברנו עד כה באסמבלי יכול לסייע לנו גם הפעם, אך ראשית כל נדרש להבין לעומק את עקרון הפעולה של ריענון המסך.

כפי שהסברתי בפרק ה' במחשבת 8, המעבד המרכזי של מחשב האטארי עובד בקצב מסחרר של למעלה ממיליון וחצי מחזורים (Cycles) בשנייה.

אתמקד עתה בפרק הזמן של שנייה אחת בה עובד המחשב. חלק הארי של אותה שנייה בודדת מוקדש לביצוע כל אותן פעולות שהמתכנת (אתה למשל) פוקד עליו, ואולם חלק קטנטן מן הזמן הזה – כלומר כמה חלקיקי שנייה – מוקדש לפעולה חשובה עד מאוד, של ציור המסך שוב ושוב על ידי ה-Antic .

חמישים פעמים במשך אותה שנייה מפריעים למעבד המרכזי במילוי משימותיו ודורשים ממנו לצאת לגיחה בהולה. דורשים ממנו לאפשר ל-Antic לצייר כהרף עין את המסך כולו (בהתאם ל-Display List כמובן), ורק משסיים משימה בהולה זו, הוא רשאי להמשיך ולבצע אל כל אותן הפעולות אשר הניח לרגע בצד.

? האם חלוקה של שנייה בודדת אחת לפרקי זמן קצרים יותר מזכירה לך דבר מה?

אם חשבת על "שניות" המחשב – צדקת בהחלט! במשך שנייה רגילה אחת מספיק שעון המחשב להתקדם כ-50 "שניות מחשב" (כזכור, הבית המונה אותן הוא בית 20).

? האם יש קשר בין שני הדברים?

מסתבר שכן!

שעון המחשב מסונכרן בתיאום מושלם עם קצב ריענון המסך, במובן זה שכל אימת שה-Antic מסיים סבב אחד של ציור המסך, באותו הרגע בדיוק מתקדם שעון שניות המחשב, שנייה אחת קדימה.

כל זה מוביל אותנו לטריק תכנותי רב חשיבות.

אם נצליח ללכוד את רגע התחלופה של "שניית מחשב" בודדת, נדע בוודאות כי ממש ברגע זה סיים גם ה-Antic את מלאכת ציור המסך, והוא ממתין "חסר מעש" עד שיידרש לשוב ולצייר המסך מחדש.

למזלנו, לכידת הרגע המיוחד הזה היא משימה פשוטה ביותר. כל שנדרש מאתנו לעשות הוא לנטר כל העת את שעון שניות המחשב, ובדיוק ברגע שבו שניית המחשב מתחלפת, לנצל את "חוסר המעש" של ה-Antic, ולמהר ולפקוד על המחשב להתקדם שלב אחד של גלילה עדינה.

דא עקה, מכיוון שלכידת "שניית מחשב" היא פעולה מהירה ביותר, נדרש גם כלי עבודה מהיר ביותר לביצוע המשימה – ודאי לא כלי עבודה של שפת הבייסיק.

נחזור אם כן לעבוד באסמבלר.

בטרם ניגש לכתוב את הרוטינה השלישית והאחרונה במשימת גלילת המסך, עלינו לדון בשימוש מורכב יותר בפקודת הבייסיק: **USR**.

בפרק י"ב במחשבת 8 הסברתי כי על מנת להפוך תוכנית שכתבנו באסמבלי לרוטינה הניתנת להרצה מתוך תוכנת בייסיק, עלינו לטעון אל תוך המחסנית לפחות נתון אחד, באמצעות פקודת האסמבלי **PLA**. עתה הגיע הזמן להתעכב מעט על עניין זה.

ישנם מקרים בהם נרצה לקרוא לרוטינה מתוך הבייסיק יותר מפעם אחת, וכל פעם לעשות בה שימוש עם ערך שונה. במקרים כאלו, נרצה לצרף לאותה קריאה לרוטינה פרמטר מסוים אשר עליו היא תפעל.

אדגים זאת באמצעות רוטינה פשוטה ביותר.

נניח שבנינו רוטינה לצביעת רקע המסך באדום, אשר נראית כך:

```
10      *=$600
20      PLA
30      LDA  #$22
40      STA  $2C8
50      RTS
```

על מנת לקרוא לרוטינה הזו מן הבייסיק (כמובן לאחר שהפכנו אותה בעזרת כתבן האסמבלר לקוד בשפת מכונה), נשתמש בפקודת הבייסיק **USR**.

עתה, נניח שברצוננו לשנות מדי פעם בפעם את צבע רקע המסך, ועדיין להשתמש ברוטינה אחת. במקרה שכזה, יועיל ביותר לקרוא לרוטינה שלנו ולשלב בקריאה הזו את ערך הצבע הנדרש.

הפקודה **USR** מאפשרת לעשות זאת, כך:

```
U=USR  (1536,34)
```

שים לב, המספר הראשון בתוך הסוגריים, הוא כידוע כתובת הזיכרון הראשונה בה יושבת הרוטינה שכתבנו ואילו המספר השני הוא פרמטר אשר הרוטינה תשתמש בו. כמובן ששני הערכים הללו מובאים בשיטת הספירה העשרונית, כיאה לסביבת הבייסיק!

כאשר אנו קוראים לרוטינה באופן הזה, נטענים מיד אל תוך המחסנית שלושה ערכים. הערך הראשון והשני נושאים יחדיו את הפרמטר (בשיטת ה-LSN, MSN). בדוגמה שלנו ייכנסו למחסנית בזה אחר זה הנתונים 34 ו-0. הערך האחרון שנכנס אל תוך המחסנית מציין את מספר הפרמטרים שעשינו בהם שימוש. בדוגמה שלנו, מכיוון שהמחסנית מאכסנת פרמטר אחד בלבד, הערך השלישי והאחרון שיוכנס למחסנית הוא: 1.

נשנה עתה את הרוטינה שלנו, כדי שתוכל אכן לעשות שימוש בפרמטר:

```
10    *=$600
20    PLA
30    PLA
40    PLA
50    STA  $2C8
60    RTS
```

שים לב, על מנת להגיע אל הנתון הרלוונטי עבורנו, הלוא הוא הפרמטר – אנו שולפים מן המחסנית בזה אחר זה את שלושת הערכים שהוכנסו אליה. לפי עיקרון ה"ב.ר.י.א" הערך האחרון שנשלוף, הוא זה שהוכנס ראשון אל המחסנית, והוא הערך 34 (או $22).

■ נסה לקרוא לרוטינה מתוך הבייסיק, בכל פעם עם פרמטר אחר, על מנת לצבוע את המסך בצבעים שונים.

בשלב זה, אגב, תוכל להבין מדוע גם כשאנו קוראים לרוטינה מן הבייסיק ללא כל פרמטר באמצעות הפקודה **USR**, עדיין עלינו לקחת בחשבון כי המחסנית נטענה בערך אחד, ועל כן עלינו לשלוף ממנה ערך אחד. זאת משום שגם במקרה הזה, המחשב טוען במחסנית ערך המציין את מספר הפרמטרים שנעשה בהם שימוש, הלוא הוא: 0.

לאחר שהבהרתי את השימוש ברוטינה עם פרמטר, נחזור לגלילה העדינה.

■ כתוב רוטינה קצרה אשר תשמש לביצוע משימת הגלילה העדינה. על הרוטינה להשתמש בפרמטר שיישלח אליה מתוך תוכנת הבייסיק; לנטר את שעון המחשב ובו ברגע שהתחלפה "שניית מחשב" לפקוד לתוך הבית האחראי על הגלילה האופקית, את ערך הפרמטר.

[תשובה 9 בעמ' 108]

■ הפוך את הרוטינה שכתבת באסמבלי, לקוד שפת מכונה והוסף אותו לתוכנת הבייסיק, על מנת לקבל סוף סוף את גלילת המסך המיוחלת.

[תשובה 10 בעמוד 108]

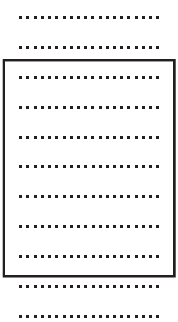
בתחילת הפרק, דימינו את עצמנו לבמאי קולנוע. בסעיף הבא נסיים את סרט הקולנוע שבו פתחנו, ונעשה זאת ברולר של כתוביות העולות כלפי מעלה.

ד. גלילה אנכית

כאשר עסקנו בנושא הגלילה האופקית הסברתי – גם באמצעות איור – כי על מנת לגלול דבר מה (טקסט, ציור, תמונה), האובייקט הנגלל חייב להיות רחב יותר מאזור התצוגה.

עיקרון זה נכון כמובן גם באשר לגלילה האנכית, אך למזלנו את היריעה האנכית קל הרבה יותר למתוח, וזאת משום שניתן להשתמש ביחידות הגרפיות מבלי לשנות את אורכן (על כל החישובים הכרוכים בכך). כל שנדרש מאתנו לעשות הוא להגדיל את אזור זיכרון המסך שלנו במספר שורות הגרפיקה (או הטקסט) שאותן אנו רוצים לגלול.

האיור להלן ממחיש כיצד תיראה יריעה אנכית רחבה:



כך לדוגמא, אם ברצוננו לגלול על מסך גרפי 2, עשרים שורות טקסט, עלינו להגדיל את זיכרון המסך, כך שיכיל בנוסף לעשר השורות המקוריות (ובנוסף לחלון הטקסט התחתון), עוד עשר שורות טקסט נוספות. ולכן במקום 200 תאי זיכרון מסך, יידרשו לנו 400 תאים (תוספת של 20X10 שהם עוד 200 תאי זיכרון).

אחר כך, בכל אחד משלבי הגלילה מעלה, עלינו לקדם את זוג הבתים המצביעים (האחד והיחיד הפעם) שב-Display List בקפיצות של 20 מקומות (כאורכה של יחידה גרפית אחת).

? ומה לגבי גלילה עדינה? האם נוכל לגלול את הטקסט כך שייראה כמו רולר של ממש?

התשובה היא כן, ובאופן דומה מאוד לגלילה האופקית העדינה.

הבית האחראי לגלילה האנכית הוא: 54277 וכל ערך שתפקוד לתוך בית זה, יגלול – פיקסל אחר פיקסל – שורות מסוימות במסך על הציר האנכי.

? וכיצד נגדיר את השורות הפוטנציאליות לגלילה האנכית העדינה?

ובכן, כל שנדרש הוא להוסיף 32 לערך קוד ההכרזה על המסך הגרפי או לערך קוד היחידה הגרפית (או בשפה "מחשבית" יותר: "להדליק בהם את הסיבית השישית, או סיבית D5").

כך למשל, אם ברצוננו לגלול גלילה אנכית עדינה את השורה הרביעית של מסך גרפי 2, נקטע את רצף החזרות 7,7,7 ובמופע השלישי נכתוב את הערך 39 (שהוא תוצאת החיבור: 32+7).

באופן דומה, אם ברצוננו לגלול גלילה אנכית עדינה את השורה הראשונה של מסך גרפי 3, נשנה את קוד ההכרזה 72, ונכתוב במקומו את הערך 104 (שהוא תוצאת החיבור 32+72).

■ נסה לכתוב בכוחות עצמך תוכנית בייסיק קצרה שתגלול מעלה על מסך גרפי מספר 2, את עשרים המילים: ONE, TWO, THREE, ..., TWENTY. לצורך בניית המסך תוכל לפתוח במסך גרפי 0 הגדול יותר מבחינת זיכרון המסך.

שים לב, כאשר אנו גוללים גלילה אנכית מסך אשר ה-Display List שלו בנוי בשיטה הסטנדרטית (ראה לעיל דימוי קטר הרכבת מושך הקרונות) החלק שניתן לגלול כלפי מעלה הוא החלק הנראה בלבד. אם תנסה להפגדיר גם את היחידות (שורות) שמתחת לאזור הנראה לניתנות לגלילה אנכית, הגלילה כולה עשויה להשתבש.

ועוד טיפ קטן: קח בחשבון כי הכתיבה ישירות אל תוך זיכרון המסך שונה מן הכתיבה על המסך עצמו (באמצעות הפקודה: **PRINT #6**). ויש להפחית 32 מערך ה-ASCI של התו הרצוי. כך למשל כדי לאכסן את האות A בתוך תא בזיכרון המסך עליך לפקוד את הערך 32, שהוא ערך ה-ASCI של האות "A" פחות 32.

[תשובה 11 בעמ' 109]

? מה דעתך על התוצאה?

ממש כמו בגלילה האופקית, גם על הציר האנכי הגלילה אמורה להיות רציפה ומהירה. ואולם, שוב (כצפוי), המסך מעט מרצד ומקפץ, כמו רוצה להזכיר לנו את מגבלת התכנות בשפת הבייסיק.

נפנה אפוא שוב לכתבן האסמבלר ונבנה רוטינה לביצוע הגלילה האנכית העדינה. במקביל, נבנה רוטינה נוספת באסמבלי לביצוע הגלילה האנכית הרגילה, וזאת על מנת ליצור תנועה חלקה ומהירה יותר.

נפתח בגלילה העדינה.

הרוטינה לביצוע גלילה אנכית עדינה דומה מאוד לרוטינה שכתבנו לביצוע הגלילה האופקית העדינה. והיא עשויה להיראות כך:

```
10     *=$600
20     PLA
30     PLA
40     PLA
50     LDX  $14
60     INX
70  LOOP
80     CPX  $14
90     BEQ  END
100     JMP  LOOP
110  END
120     STA  $D405
130     RTS
```

■ הפוך את שורות הקוד הללו לרוטינה בשפת מכונה ושבץ אותה בתוך תוכנת הבייסיק. אך לפני כן, אמליץ לך לייצר גם עבור הגלילה האופקית הרגילה רוטינה בשפת מכונה. אל תוך הרוטינה הזו הכנס מקטע קוד אשר יאפס את הגלילה האנכית העדינה. כאשר תשלב את שתי הרוטינות בתוכנת הבייסיק, תתקבל תנועת גלילה רציפה וחלקה.

[תשובות 12 ו-13 בעמ' 110]

סיכום

בפרק זה למדנו להרחיב את היריעה כדי להפוך את מסך הטלוויזיה לנגלל.

הגלילה (הן האנכית והן האופקית) נחלקת לשתי רמות:

(1) גלילה "גסה" אשר מושתתת על מניפולציה על ה-Display List: ככל שנשנה את כתובתם של הבתים המצביעים על אזורים רלוונטיים בזיכרון המסך, נוכל ליצור אצל הצופה **תחושה** של תזוזה.

(2) גלילה "עדינה" אשר מבוססת על 2 בתים ייעודיים:

- בית 54276 אחראי על הגלילה העדינה האופקית.
  על מנת לגלול על הציר האופקי יחידה גרפית, יש להדליק את הסיבית החמישית שלה (כלומר להגדיל ב-16 את קוד ההכרזה או את קוד היחידה הגרפית)
- בית 54277 אחראי על הגלילה העדינה האנכית.
  על מנת לגלול על הציר האנכי יחידה גרפית, יש להדליק את הסיבית השישית שלה (כלומר להגדיל ב-32 את קוד ההכרזה או את קוד היחידה הגרפית).

דע לך כי על מנת שתוכל לגלול גרפיקה הן על הציר האופקי והן על הציר האנכי, עליך להדליק גם את הסיבית החמישית וגם את הסיבית השישית, כלומר להגדיל ב-48 את קוד ההכרזה או את קוד היחידה הגרפית.

פרק ג
פסיקת מתווה התצוגה
(Display List Interrupt)

פרק זה העוסק בהפרעות (או פסיקות) של מתווה התצוגה – Display List Interrupts או בקיצור DLI – מציג את אחת היכולות החזקות ביותר המובְנות במחשב האטארי. כדי להבין במה מדובר, נפתח ברקע תיאורטי קצר.

ודאי תופתע לגלות כי כאשר אתה מתבונן על מסך הטלוויזיה שלך, אתה מתבונן למעשה על **נקודה בודדת אחת** – על אלומה דקה של אלקטרונים אשר "מופגזת" ללא הרף על מסך הטלוויזיה באמצעות "אקדח אלקטרונים". "אקדח האלקטרונים" הזה מתרוצץ הלוך ושוב במהירות מסחררת מראש המסך ועד לתחתיתו וגורם למוח שלך להאמין כי מדובר בתמונה רציפה.

דע לך כי ישנו אזור מיוחד בזיכרון המחשב אשר אחראי לצבעה של אותה אלומת אלקטרונים, ואזור זה הוא טווח הבתים 53266 עד 53274.

ניקח לדוגמה את הבית 53272 (או $D018). בית זה קובע כיצד תתנהג אלומת האלקטרונים כשהיא מתרוצצת באזור המסך (באזור הריבוע הכחול).

■ עבור לסביבת אסמבלר והקלד את הקוד הקצר הבא:

```
10      *=$600
20 LOOP
30      INX
50      STX  $D018
60      JMP  LOOP
```

■ הרץ את הקוד וראה את מופע הפסים הצבעוני.

כאשר אנו פוקדים על המחשב לשנות שוב ושוב, במהירות הבזק, את צבעה של אלומת האלקטרונים אנו יכולים לקבל מושג על מהירות תנועתה.

קצב ריענון המסך, כפי שאתה כבר יודע הוא 50 פעמים בשנייה (או 50 הרץ) וחישוב פשוט מראה כי לאותו "אקדח אלקטרונים" לוקח 0.02 שניות (או 20 מילי-שניות) להשלים סיבוב אחד של ציור המסך כולו, כלומר להשלים את צביעתם של 192 פסים אופקיים.

פרקי זמן בסדר הגודל הזה נראים לנו בתפיסתנו האנושית "הרף עין" ממש, אך במונחיו של המעבד המרכזי אלו פרקי זמן ארוכים למדיי, פרקי זמן המאפשרים לו לעקוב אחר אותו "אקדח אלקטרונים" תזזיתי ולהציב לו "באמצע הדרך" שינויים בפרמטרים מסוימים, לפי בקשתנו. זאת ועוד, המעבד המרכזי הזריז והדייקן יודע גם לחזור בדיוק מוחלט על פעולתו זו פעם אחר פעם – 50 פעמים בשנייה, ולהציב ל"אקדח האלקטרונים" שוב ושוב את אותו השינוי המסוים שביקשנו, ובאותו המקום בדיוק.

הכנסת שינויים בפרמטרים של המסך, במהלך עבודתו הרציפה של אותו "אקדח אלקטרונים" נקראת "הפרעה" (או "פסיקה"), ועל מנת להשחיל את אותה הפרעה במקום המתאים ובתזמון המתאים, נדרשים כלי עבודה מהירים ביותר. סביבת האסמבלר היא סביבת עבודה מצוינת למטרה זו, אך בטרם נעבור לכתיבת הקוד, עלינו להבין תחילה את עקרונות הפרעת ה-DLI.

התהליך מתחיל כאשר ה-Antic נתקל בהוראה עם פקודת הפסיקה. צורת הפקודה היא הוספת 128 לערך קוד ההכרזה על המסך הגרפי או לערך קוד היחידה הגרפית (או בשפה "מחשבית" יותר: "הדלקת הסיבית השמינית, או סיבית D7"). ה-Antic בתגובה ממתין עד אשר "אקדח האלקטרונים" יסיים

את סריקת השורה שאותה התחיל, ואז בודק אם הפרעות DLI הוגדרו בכלל כאפשריות.

אם הפרעות מסוג DLI לא הוגדרו כאפשריות, ה-Antic מתעלם מן הפקודה ושב לפעולותיו השוטפות. אם הפרעות מסוג DLI כן הוגדרו כאפשריות, ה-Antic ניגש לבצע את הרוטינה הייחודית שכתבנו.

את הרוטינה נוכל לשמור בזיכרון כרגיל בדף מספר 6, אך הפעם לא נקרא לה באמצעות פקודת הבייסיק **USR**, אלא נמסור למחשב את כתובת הזיכרון בה היא יושבת. זוג הבתים המצביעים אשר נושאים את כתובת הזיכרון בה שמורה רוטינת ה-DLI הם: 512 ו-513.

עם סיום ביצוע פעולת הרוטינה, "אקדח האלקטרונים" ממשיך במשימתו עם הפרמטרים החדשים שקיבל, ואילו המעבד המרכזי שב לביצוע פעולותיו השוטפות, עד אשר ייקרא שוב לביצוע אותה המשימה בדיוק – 20 מילי-שניות מאוחר יותר...

עד כאן הרקע התיאורטי, ועתה לכתיבת הרוטינה!

הרוטינה שנכתוב חייבת להיות קצרה ומהירה; עליה לשנות אך ורק אספקטים הקשורים לתצוגת המסך; ואסור לה להשאיר עקבות (אם אינך מבין את הכוונה, עיין בפרק ט' במחשבת 8, אשר עוסק בהפרעות מסוג NMI). זכור כי הרוטינה תפעל "מאחורי הקלעים" ללא הפסק, ולכן באחריותנו לדאוג כי כל רגיסטר שנעשה בו שימוש ברוטינה שלנו, יחזור – עם היציאה מן הרוטינה – לערכו ההתחלתי.

נפתח בדוגמה פשוטה – צביעת חלקו התחתון של המסך בצבע ורוד.

■ עבור לסביבת אסמבלר והקלד את שורות הקוד הבאות:

```
10    *=$600
50    LDA #$50
60    LDX #$58
80    STA $D017
90    STX $D018
```

אם תריץ כעת את הקוד שכתבת באסמבלי (נסה!), תגלה שהמסך אינו משנה את צבעו, אלא רק מהבהב לרגע בצבע ורוד ושב מיד לקדמותו!

אילו רצינו לשנות את צבע המסך, היינו משתמשים באחד מן הבתים בטווח שבין 712-704 – בתים אשר ודאי כבר מוכרים לך כאחראים על שינוי צבעי המסך. ואולם אנו השתמשנו בטווח שונה לחלוטין של בתים: 53274-53266, בתים השולטים על צבע "אלומת האלקטרונים" הסורקת את המסך.

? מה ההבדל ביניהם?

ובכן, במסגרת חוברת לימוד זאת לא אכנס להסברים מעמיקים על מערכת ההפעלה של מחשב האטארי, אלא רק אבהיר כי הבתים בטווח 53274-53266, הם **היחידים** המנהלים את מנגנון הצבע של ה-Antic. הבתים בטווח 712-704 נועדו אך ורק לצרכים תכנותיים – להקל על משתמשי המחשב בכלל והמתכנתים בבייסיק בפרט, לשנות את צבעי המסך מבלי להידרש לכתוב רוטינות שיפעלו על ה-Antic. דע לך כי כל נתון שאנו מכניסים אל תוך אחד הרגיסטרים בטווח הבתים 712-704 מועתק בדיוק ברגע ריענון המסך אל תוך הרגיסטר המקביל לו בטווח הבתים 53274-53266. מבחינה זו טווח הבתים 712-704 משמש למעשה כצל (shadow) לבתים בכתובות הזיכרון הגבוהות יותר.

בכל אופן, כשאנו כותבים רוטינה משל עצמנו המבצעת מניפולציות על ה-Antic עצמו, לעולם לא ניגש אל בתי ה"צל" אלא נפקוד היישר אל תוך כתובות הזיכרון הגבוהות.

נשוב עתה אל הרוטינה ונוסיף לה את הפקודות המבטיחות שההפרעה תתבצע בלא להותיר שום עקבות. הרגיסטרים שאנו עושים בהם שימוש הם האוגר (רגיסטר A) ורגיסטר X, ולכן נכניס את שניהם מבעוד מועד בזה אחר זה אל תוך המחסנית.

```
20      PHA
30      TXA
40      PHA
```

לאחר שהרוטינה סיימה את פעולתה (החלפת הצבע) נדאג לשלוף את ערכי הרגיסטרים המקוריים ולהשיבם לקדמותם.

```
100     PLA
110     TAX
120     PLA
```

נותר לנו להכניס לרוטינה עוד שתי פקודות. ראשית, עלינו להבטיח כי כל שינוי שנכניס בפרמטרים של הצבע (במקרה שלנו הפיכתו מכחול לוורוד) יהיה מסונכרן היטב עם "אקדח האלקטרונים" המייצר את אלומת האלקטרונים. אחרת, ייראה ריצוד של צבע איפשהו במרכז המסך.

בית הזיכרון 54282 ($D40A) אמון על משימה זו בדיוק. כל ערך שנפקוד לתוכו (בין 0 ל-255), יבטיח כי אותו "אקדח אלקטרונים" יסיים את השורה האופקית אשר אותה התחיל לסרוק, בטרם ייכנסו לתוקף השינויים

שביקשנו. נהוג לכנות בית זה WSYNC (קיצור המילים: Wait for horizontal SYNChronization – "המתן לסריקה האופקית").

נוסיף אפוא את שורת הקוד לרוטינה:

```
70      STA  $D40A
```

לבסוף, עלינו להוסיף פקודה שמורה למעבד המרכזי לחזור מן הרוטינה, כלומר לסיים את ההפרעה. פקודה זו היא הפקודה (היחידה) אשר לא עברנו עליה ביחידת הלימוד "מחשבת 8" והיא: RTI (קיצור המילים: ReTurn from Interrupt).

נוסיף גם אותה לרוטינה שלנו:

```
130     RTI
```

סיימנו לכתוב את רוטינת ה-DLI. עתה נותר לנו להפעילה!

את זאת נעשה כהרגלנו מתוך הבייסיק.

■ הפוך את הרוטינה שכתבת לקוד שפת מכונה, רשום לפניך את ערכי הקוד (העשרוניים כמובן) ועבור לסביבת הבייסיק.

ראשית, נבחר את השורה על המסך בה נרצה להכניס את ההפרעה שלנו. נניח שאנו רוצים להכניס את ההפרעה בדיוק במרכז המסך. מכיוון שישנן 24 יחידות גרפיות על מסך גרפי 0, ההפרעה צריכה להיכנס במקום ה-11 (כזכור, הפרמטרים משתנים תמיד שורה אחת אחרי ההפרעה).

היחידה הגרפית הראשונה על המסך נקבעת כזכור בהכרזה על סוג המסך הגרפי ועל כן, במופע העשירי של קוד הגרפיקה, נוסיף את הסיבית השמינית (D7) ונחליף את הערך 2 בערך 130.

```
10 DLIST=PEEK(561)*256+PEEK(560)
20 POKE DLIST+15,2+128
```

עתה, נטען את הרוטינה המקודדת בשפת מכונה אל תוך הדף מספר 6 בזיכרון, ואת הכתובת בה יושבת הרוטינה נכניס אל תוך זוג הבתים המצביעים: 512, 513

```
30 READ A: IF A=-1 THEN 60
40 POKE 1536+I,A
50 I=I+1: GOTO 30
60 POKE 512,0:POKE 513,6
100 DATA 72,138,72,169,80,162,88,141,
10,212,141,23,208,142,24,208,104,170,
104,64,-1
```

לבסוף, עלינו להודיע למחשב כי אנו מאפשרים הכנסת הפרעות ("פסיקות") מסוג DLI.

בית 54286 אמון על הפרעות מסוג NMI (אשר DLI היא אחת מהן). אם אינך זוכר מהי הפרעה מסוג NMI, עיין בפרק ט' ב"מחשבת 8".

על מנת להתיר למחשב להגיב להפרעות מסוג DLI נפקוד לתוך בית זה את הערך: 192.

```
70 POKE 54286,192
```

■ הרץ את תוכנת הבייסיק וראה כיצד המסך משנה את הנראות שלו. אפיון זה של המסך יישאר בתוקף עד אשר תפקוד על המחשב: **GR.0**, או עד אשר תלחץ על המקש RESET.

עמידה בלוחות זמנים

בעמודים הקודמים פגשנו את הבית 54282 ($D40A), המכונה גם "WSYNC", וראינו כי בכל פעם שנפקוד ערך כלשהו לתוכו, המעבד המרכזי "יקפא" לרגע וימתין עד אשר "אקדח האלקטרונים" יסיים את השורה האופקית אשר אותה התחיל לסרוק. במובן זה, אותו "WSYNC" משמש מעין "רמזור אדום" המווסת את פעילותו של המעבד המרכזי.

? מה יקרה אם לא נעשה שימוש ב-WSYNC ברוטינות ה-DLI שנכתוב?

הבה ננסה.

■ חזור אל הרוטינה האחרונה שכתבת (אשר צובעת את חציו התחתון של המסך בוורוד) ומחק את הפקודה הטוענת ערך לתוך ה-WSYNC. הפוך את הרוטינה שוב לקוד בשפת מכונה וטען אותה לתוכנת הבייסיק.

שים לב. אם אתה כבר חש בטוח ברמת שליטתך בשפת המכונה, תוכל לדלג לעתים על שלב עדכון קוד האסמבלי, לגשת במישרין אל קוד שפת המכונה השמור כבר בתוכנת הבייסיק, ולשנותו בזהירות (!)

כל שעליך לעשות הוא לאתר את הרצף 141,10,212 (שכן הנמוניק **STA** בשפת האסמבלי שקול לפקודה 141, ואילו הבית 54282 ייוצג בשיטת ה-MSD, LSD כך: 10 ,212). לאחר שאיתרת את הרצף הזה, תוכל למחוק אותו ללא חשש, וזאת משום שהרוטינה **אינה כוללת פקודות דילוג פנימיות.**

בכל אופן, שורת הבייסיק המתקבלת בסופו של דבר היא זו:

```
100  DATA  72, 138, 72, 169, 80, 162, 88, 141,
23, 208, 142, 24, 208, 104, 170, 104, 64, -1
```

■ הרץ את התוכנה וראה את הריצוד סמוך לקצהו הימני של המסך.

דע לך כי זמן הסריקה של שורה בודדת אחת (קרי הזמן אשר לוקח לאותו "אקדח אלקטרונים" לעבור מקצהו השמאלי ביותר של המסך ועד לקצהו הימני ביותר) הוא 114 מחזורים (Cycles),[5] וגם אם ניקח בחשבון כי בחלק מן הזמן הזה המעבד המרכזי עסוק במשימות חיוניות אחרות, עדיין – כאשר אנו פוקדים על המעבד המרכזי להמתין לסיום שורת הסריקה, יכולים "להתבזבז" כמה עשרות מחזורים באותה המתנה.

כמובן שעבור יציאה בודדת להפרעת DLI פשוטה (כמו זו שבנינו), זמן ההמתנה של המעבד המרכזי הוא זניח ולא מורגש כלל,[6] אך ככל שרוטינות ה-DLI הולכות ומסתבכות המשתמש עשוי להרגיש באיטיות בתגובת המחשב. לכן, בבואנו לבנות רוטינות DLI, כדאי לקחת בחשבון גם את שיקולי הזמן.

נחלק את זמן פעולת רוטינת ה-DLI לשלושה מקטעים:

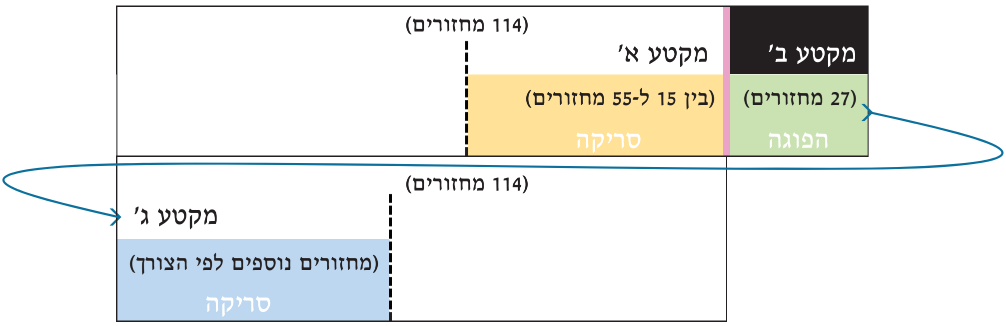
- מקטע א' – מן הרגע שבו פגש המעבד המרכזי את פקודת היציאה להפרעה (ב-Display List) ועד אשר בוצעה פקודת ה-WSYNC. במשך פרק זמן זה מסיים "אקדח האלקטרונים" לסרוק שורה שלמה אחת.
- מקטע ב' – מן הרגע שבו סיים "אקדח האלקטרונים" לסרוק את השורה, ועד לרגע שבו הוא מתחיל לסרוק את השורה הבאה. במשך פרק הזמן הזה "אלומת האלקטרונים" כבויה לחלוטין.

---

5 לשם השוואה, ביצוע פקודת האסמבלי LDA למשל, אורך בין 2 ל-4 מחזורים.
6 זכור כי במהלך שנייה אחת מבצע המעבד המרכזי למעלה ממיליון וחצי מחזורים(!)

- מקטע ג' – מן הרגע שבו "אקדח האלקטרונים" התחיל לסרוק את השורה **הבאה** ועד אשר רוטינת ה-DLI שלנו מסתיימת והמעבד המרכזי חוזר לבצע את משימותיו השוטפות.

אציג את שלושת המקטעים באיור הבא:



משכו של מקטע א' (המסומן באיור בצבע צהוב) קטן משמעותית מ-114 מחזורים, וזאת משום שהמעבד המרכזי חייב לסיים משימות אחרות בטרם יתפנה להריץ את רוטינת ה-DLI. למעשה, המעבד המרכזי מתפנה לצרכינו התכנותיים אחרי כמה עשרות מחזורים, ובסיכומו של דבר עומדים לרשותנו בסך הכול בין 15 מחזורים (ביחידות גרפיות מסוג: 0, 7, 8) ועד ל-55 מחזורים (ביחידה הגרפית הריקה לחלוטין).

את מקטע א' נייחד לכל הפקודות אשר **אינן** נוגעות בפרמטרים של תצוגת המסך, ובראש ובראשונה – הפקודות הטוענות לתוך המחסנית את הרגיסטרים של המעבד המרכזי. בסיומו של מקטע א' תבוא פקודת ה-WSYNC, כך שאם נותר עוד פרק זמן פנוי עד לסיום סריקת השורה, הוא "יתבזבז" במנוחה של המעבד המרכזי.

משכו של מקטע ב' (המסומן באיור בצבע ירקרק) הוא 27 מחזורים, ופרק הזמן הזה פנוי כמעט במלואו להמשך הרצת הרוטינה. מקטע ב' ייוחד לכל הפקודות **המשנות את הפרמטרים של תצוגת המסך** (כגון שינוי צבע). כל עוד "אקדח האלקטרונים" שרוי בהפוגה, אפשר לשנות פרמטרים אלו ללא חשש לריצוד.

משכו של מקטע ג' (המסומן באיור בצבע תכלת) תלוי במורכבותה של הרוטינה שבנינו. גם הפקודות שבמקטע זה, אסור להן לשנות פרמטרים של תצוגת המסך, וזאת משום ש"אקדח האלקטרונים" שוב עסוק בסריקה של השורה (הבאה בתור).

לפני שאחתום את סעיף זה, אדגים את לוחות הזמנים של ביצוע רוטינת ה-DLI האחרונה שבנינו – בחלוקה לפי שלושת המקטעים. לצד כל פקודת אסמבלי, מוצג זמן הביצוע שלה (ביחידות של מחזורי מחשב):

10 *=$600		
20 PHA	3 מחזורים	12 מחזורים
30 TXA	2 מחזורים	
40 PHA	3 מחזורים	
50 LDA #$50	2 מחזורים	
60 LDX #$58	2 מחזורים	
70 STA $D40A	4 מחזורים	WSYNC
80 STA $D017	4 מחזורים	24 מחזורים
90 STX $D018	4 מחזורים	
100 PLA	4 מחזורים	
110 TAX	2 מחזורים	
120 PLA	4 מחזורים	
130 RTI	6 מחזורים	

שים לב, מכיוון שרוטינת ה-DLI פשוטה ומהירה, היא מסתיימת כבר במקטע ב׳ ולא גולשת אל זמן סריקת השורה הבאה.

קשת הצבעים

בתחילת הפרק הקלדת תוכנה קצרה באסמבלר על מנת לחזות במהירות התזזיתית של ״אקדח האלקטרונים״. כאשר קידמנו כל העת את ערכו של הבית האחראי על צבעה של האלומה, קיבלנו מופע צבעוני של פסים.

■ חזור אל אותה רוטינה והוסף לה שורת קוד הפוקדת ערך כלשהו ל-WSYNC :

```
40    STX  $D40A
```

הרץ את הרוטינה וראה את קשת הצבעים המתנועעת על המסך.

? האם ניתן לשלב אפקט מרהיב שכזה בתוך רוטינת DLI ?

בהחלט!

ראשית, נבנה רוטינה שהופכת את מחציתו של המסך לפריסה סטאטית של הגוונים המרהיבים.

בסעיף הקודם תזמנתי את רוטינות ה-DLI והראיתי כי על מנת להימנע מאיטיות בתגובת המחשב, עלינו להקפיד להגביל את מספר הפקודות במקטע הראשון (לפני ה-WSYNC) ולוודא שכל הפקודות אשר משנות את

הפרמטרים של המסך לא יחרגו מן המקטע השני (כלומר מפרק הזמן בו "אקדח האלקטרונים" שרוי במנוחה).

ואולם, לפעמים נסכים לשלם את מחיר האיטיות כדי לקבל אפקט מרהיב על פני המסך.

למעשה, במקום לסיים את הרוטינה בטווח של מקטע ב', נוכל לגלוש לא רק אל תוך מקטע ג' (כלומר אל תוך סריקת השורה הבאה), אלא גם אל תוך שורת הסריקה השלישית, הרביעית, וכולי. ובלבד, שבכל פעם ופעם נקפיד על ה-WSYNC. במילים אחרות, נוכל לבנות רוטינת DLI שתתפרש על פני כמה וכמה שורות סריקה, כך שבכל אימת ש"אקדח האלקטרונים" שרוי במנוחה, יבוצע שינוי בפרמטר של המסך, ובכל אימת ש"אקדח האלקטרונים" מתחיל לסרוק את השורה הבאה, המעבד המרכזי "יקפא" ויניח לסריקת השורה להסתיים.

המחיר לכך הוא כאמור עיכוב בעבודת המעבד המרכזי.

? עד כמה העיכוב הזה חמור?

נחשב.

נניח שבנינו רוטינה המתפרשת על 177 קווי סריקה (כלומר רוטינה בה פוקדים 177 פעם את פקודת ה-WSYNC). אם משך אורכה של כל המתנה כזאת הוא בממוצע 30 מחזורים, הרי שעל מנת למלא את כל 177 השורות באפקט הצבעוני, מצטברים יחד למעלה מ-5000 מחזורי המתנה, שהם כ-3 מילי-שניות.

אולי זה לא נשמע הרבה, אך אם ניקח בחשבון שזמן הסריקה של מסך שלם ללא הפרעות הוא כ-20 מילי-שניות, הרי שמדובר על האטה של כ-15% – וזה בהחלט מורגש.

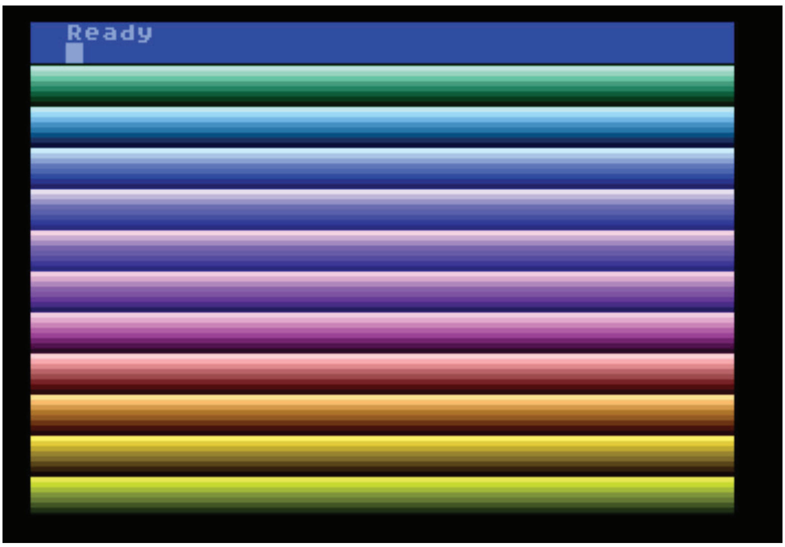
אחרי שהסברתי את העיקרון העומד מאחורי הרוטינה, נעבור לשלב התכנות.

- בנה רוטינת DLI אשר תייצר 177 קווי סריקה משתנים, בסדרת ערכים יורדת. הערך הראשוני הוא 177, והוא ילך וירד בלולאה מותנית, עד לערך 0.

[תשובה 14 בעמ' 111]

- הפוך את קוד האסמבלי לקוד בשפת מכונה והכנס את הערכים אל תוך תוכנת הבייסיק האחרונה שכתבת. לפני שתריץ את התוכנה, עדכן את שורה 20 בתוכנת הבייסיק על מנת שאפקט הקשת יתחיל מהשורה השלישית של המסך (ולא מהאמצע).

התבונן באפקט הצבעוני והתרשם גם ממהירות התגובה של המחשב.



הרוטינה הבאה שנכתוב היא הפיכת חציו התחתון של המסך לאנימציה של קשת הצבעים. לשם כך, נוכל להיעזר בבית 20 (או $14) המונה את שניות המחשב. מכיוון שתדירות ריענון המסך היא כזכור 50 פעמים בשנייה (כקצב תחלופת שניות המחשב), נוכל להיות בטוחים שכל אימת שהמעבד המרכזי יוצא להפרעה, הערך השמור בבית 20, גדל בדיוק ביחידה אחת.

כל שנותר לנו אפוא, הוא להציב את ערך הבית הזה בתוך הבית האחראי על צבעה של אלומת האלקטרונים.

■ בנה רוטינת DLI אשר תייצר 177 קווי סריקה משתנים, בסדרת ערכים יורדת. הערך הראשוני יהיה זהה לערכו של בית 20, והוא ילך ויקטן בלולאה מותנית, עד לערך 0.

[תשובה 15 בעמ' 112]

הפרעות חוזרות ונשנות

? האם אפשר להציב בתוך ה-Display List יותר מפקודת הפרעה אחת, כלומר לצאת להפרעה מסוג DLI יותר מפעם אחת בזמן ציור המסך?

התשובה היא כן – בזמן סריקת המסך ניתן בהחלט לצאת כמה וכמה פעמים להפרעה, אך תמיד תהא זו **אותה הפרעה**, קרי אותה רוטינת DLI.

אשר על כן, על מנת לשנות פרמטרים בכמה נקודות על המסך, נצטרך לבסס את הרוטינה שלנו על מידע מתחלף **חיצוני לה**.

למעשה בסעיף הקודם נעזרנו במידע מתחלף שכזה – הלוא הוא ערכו של הבית 20 האחראי על שניות המחשב. להלן נבחן עוד כמה שיטות תכנות.

1. ניטור "אקדח האלקטרונים"

דע לך כי המחשב יודע לעקוב כל העת אחר אותה "אלומת האלקטרונים". הבית האמון על מניית השורות המצוירות על המסך הוא 54283 ($D40B). בית זיכרון זה יודע בכל רגע נתון היכן על המסך מצוי אותו "אקדח האלקטרונים", וכשאר אנו יוצרים להפרעה מסוג DLI אפשר לשלוף את המידע על מיקום האלומה, ולהתבסס על ערך זה על מנת לבצע שינויים מתחלפים בפרמטרים של המסך.

אדגים...

■ עבור לסביבת האסמבלר, והקלד את הקוד הקצר הבא:

```
10    *=$600
20    PHA
30    LDA  $D40B
40    STA  $D40A
50    STA  $D018
60    PLA
70    RTI
```

שים לב. בשורה 30, אנו שולפים את הערך המציין את מיקומה של "אלומת האלקטרונים" על המסך. בשורה 50 אנו מציבים את הערך הזה בתוך הבית האחראי על צבע המסך. ואילו שורה 40 היא כידוע פקודת ה-WSYNC.

- הפוך את קוד האסמבלי לקוד בשפת מכונה; חזור לסביבת הבייסיק, וכתוב תוכנה קצרה שתצא מתוך מסך גרפי מספר 0, 23 פעמים להפרעה שכתבת.

[תשובה 16 בעמ' 113]

- עתה, הוסף שתי שורות קוד לרוטינה אשר יהפכו את המסך לרצף של: שורה כהה, שורה בהירה, וחוזר חלילה. רמז: עיין בפרק י"א במחשבת 8.

[תשובה 17 בעמ' 113]

2. שימוש בבית נוסף, חיצוני לרוטינה

בסוף פרק ו' במחשבת 8 הצגתי את העיקרון התכנותי לפיו "מניחים בצד" נתון מסוים, על מנת לשוב ולהשתמש בו בהמשך (אם אינך זוכר, חזור למחשבת 8 ורענן את זיכרונך). עיקרון זה הוא כידוע, גם עקרון פעולתה של מחסנית המחשב (Stack).

כאשר אנו בונים רוטינות DLI, המחסנית משמשת אותנו כזכור כדי לצאת להפרעה "מבלי להותיר עקבות", ולשם כך אנו טוענים לתוכה (ואחר כך גם שולפים מתוכה) את הרגיסטרים A, X, Y (כולם או חלקם). בכל הנוגע לבניית רוטינות DLI, כדאי אפוא לייחד את המחסנית אך ורק למטרה הזו.

יחד עם זאת, נוכל להיעזר בבית נוסף, בתוך דף 6 למשל, ולשמור בתוכו מידע שיסייע לנו לשנות שוב ושוב פרמטרים של התצוגה, בכל יציאה ויציאה להפרעה.

אפתח בדוגמה פשוטה.

נניח שאנו רוצים לחלק את המסך לשלושה צבעים: אדום, צהוב, ירוק (כצבעי הרמזור, או כצבעי דגל בוליביה).

נוכל להכניס 3 פקודות הפרעה לתוך ה-Display List (למשל בבית ה-6, בבית ה-14 ובבית ה-21) ולבנות רוטינה שתכלול את שלושת הגוונים הרצויים. את המעבר בין גוון לגוון נוכל לעשות באמצעות שימוש בבית שהוא חיצוני לרוטינה, אך עדיין בגבולות דף 6.

■ נסה לבנות בעצמך את הרוטינה אך בכל מקרה, עיין גם בהסבר שלי.

ראשית, נבנה את השורות הסטנדרטיות הטוענות את המחסנית ושולפות ממנה את שני הרגיסטרים שנעשה בהם שימוש:

```
10    *=$600
20    PHA
30    TXA
40    PHA
140     PLA
150     TAX
160     PLA
170     RTI
```

נבחר בית חיצוני לרוטינה $628 – בית שיוכל לקבל אחד משלושה ערכים: 0, 1 או 2. במקביל, נבנה בסיס מידע לשלושת ערכי הגוונים הרצויים. לצורך כך ניעזר באוגר וכן בתווית (אם אינך מכיר את צורת השימוש הזו בתווית, עיין בפרק ח' במחשבת 8):

```
50  LDX  $628
60  LDA  DATA,X
220 DATA
230     .BYTE  $44,$FE,$C6
```

נוסיף את פקודת ה-WSYNC ואת שינוי צבע האלומה:

```
70    STA  $D40A
80    STA  $D018
```

בשלב זה נוסיף את שורות הקוד המקדמות את ערכו של הבית $628 ונוודא שהוא אכן נמצא בטווח שבין 0 ל-2:

```
90    INC  $628
100     LDX  $628
110     CPX  #$3
120     BEQ  IPUS
130  END
180  IPUS
190     LDX  #$0
200     STX  $628
210     JMP  END
```

■ בנה תוכנת בייסיק שתטען רוטינה זו למקומה המתאים בזיכרון ותיצור את המסך המחולק לשלושה מקטעים צבעוניים.

[תשובה 18 בעמ' 114]

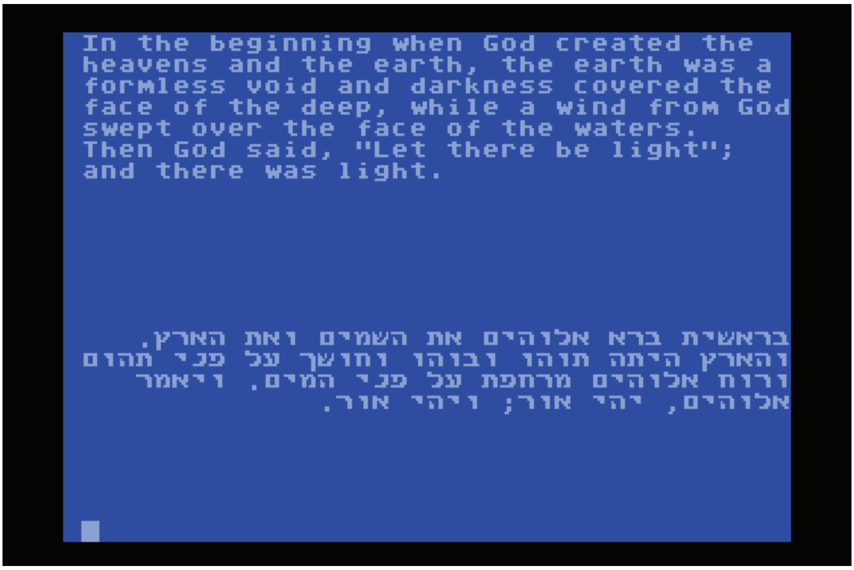
מערכי סימנים מרובים

עד כה למדנו לנצל את אופי פעולת החומרה על מנת להוסיף אפקטים צבעוניים על המסך. למעשה, ישנם דברים נוספים שנוכל "להלביש" על אותו ריענון מסך מחזורי. אחד מהם הוא שילוב מערך סימנים נוסף (או יותר) באותו המסך.

במחשבת מס' 6 למדנו לעצב מערך סימנים חדש – את אותיות האלף-בית בעברית. לצורך הכתיבה בעברית, נאלצנו לוותר על האותיות האנגלית הקטנות ובמקומן מיקמנו את האותיות העברית. לדוגמא במקום האות "a" מיקמנו את האות "ש".

כעת, בעזרת הידע שצבענו בנושא ה-DLI נוכל לכתוב על אותו מסך גם בעברית, וגם באנגלית (באותיות גדולות וקטנות גם יחד).

■ ודא כי יש בידיך תוכנה שמאפשרת כתיבה בעברית, ועבור לעבוד באסמבלר.



על מנת להפוך את המסך שלנו למסך עם שני סטים של מערכי סימנים נזדקק לפקודת הפרעה אחת, ולרוטינה קצרה אשר בה נפקוד על המחשב לעבור לסט אחר של סימנים. אך לפני כן הסבר קצר.

בפרק הקודם שעסק בפרמטרים של צבעי המסך, הסברתי את ההבדל בין הבתים האחראים לצבעה של "אלומת האלקטרונים", לבין הבתים הנמוכים בזיכרון (shadow) אשר מקלים על המתכנת לשנות את צבעי המסך.

גם במקרה של מערך הסימנים, נבחין בין הבית הגבוה בזיכרון המשמש את מערכת ההפעלה **ומיידע** אותה היכן נמצא מערך הסימנים, לבין הבית הנמוך בזיכרון (shadow) המאפשר לך, המתכנת, **לשנות** את מיקום מערך הסימנים. הבית 756 המוכר לנו כבר ממחשבת 6 הוא ה-shadow, ואילו הבית המיידע את מערכת ההפעלה היכן מצוי מערך הסימנים הוא 54281 (או: $D409).

ברוטינת ה-DLI שנכתוב, נשתמש כמובן אך ורק בבית הגבוה בזיכרון, שכן אין ברצוננו לשנות באופן קבוע את מערך הסימנים על המסך, אלא רק להחליפו באופן זמני, באותו הרגע בו יוצא המעבד המרכזי להפרעה.

■ כתוב רוטינת DLI קצרה שתיידע את המחשב במיקום המחדש של מערך הסימנים. הפוך את הרוטינה לקוד בשפת מכונה והטמע אותו בתוכנת בייסיק.

אל תשכח להריץ את תוכנת הבייסיק כאשר מערכך סימני העברית כבר יושב בזיכרון.

[תשובה 19 בעמ' 114]

דע לך כי חלוקת המסך למקטעים שונים המכילים מערכי סימנים שונים היא טכניקה נפוצה בתכנות משחקים. כך, נוכל לעצב מערכי סימנים שישמשו כדמויות (או חלקי דמויות), אשר יתנועעו בחלקו העליון של המסך, ואילו בחלקו התחתון להשתמש במערך אחר של סימנים.

סיכום

בפרק זה למדנו לשלוט בחומרה ולנצל את אופי פעולתה לצרכים תכנותיים. אלומת האלקטרונים התזזיתית היוצרת **במוחנו** את **אשליית** תמונת המסך היא מהירה ביותר, אך עדיין פעולתו של המעבד המרכזי מהירה ממנה עשרות מונים. פער זה מאפשר לנו – בתוך סביבת העבודה המהירה של שפת המכונה – להכניס שוב ושוב מניפולציות תוך כדי סריקה בודדת של המסך.

ה-Antic נדרש כזכור 50 פעמים בשנייה לרענן את המסך ולציירו מחדש. בכל אימת שהוא יוצא למשימה בהולה זו, המעבד המרכזי מפסיק לרגע את פעולתו וכל משאבי המחשב מופנים למשימת ציור המסך. אותן הפסקות מובנות נקראות בשפת המחשב "פסיקות" או "הפרעות" ועל אחת מהן – פסיקת מתווה התצוגה (Display List Interrupt) – הקדשתי את הדיון בפרק זה.

עוד למדנו בפרק זה על ההבל בין בתי זיכרון גבוהים המשמשים את מערכת ההפעלה של המחשב, לבין בתי זיכרון נמוכים (shadows) המאפשרים למתכנת לשנות פרמטרים שונים של המסך.

בפרק הבא נעסוק בפסיקה (הפרעה) חשובה נוספת – פסיקת המרווח האנכי (Vertical Blank Interrupt).

פרק ד
פסיקת המרווח האנכי
(VERTICAL BLANK INTERRUPT)

בפרק הקודם עקבנו אחר התנהגותו של "אקדח האלקטרונים" המתנועע שורה אחר שורה, לרוחבו של המסך. אותו "אקדח האלקטרונים" שרוי למשך פרק זמן מסוים במשימת ציור, וביתרת הזמן הוא למעשה כבוי ומותיר למתכנת פנאי להכניס הפרעות (פסיקות) קצרות משלו – הפרעות אשר משנות פרמטרים שונים של התצוגה.

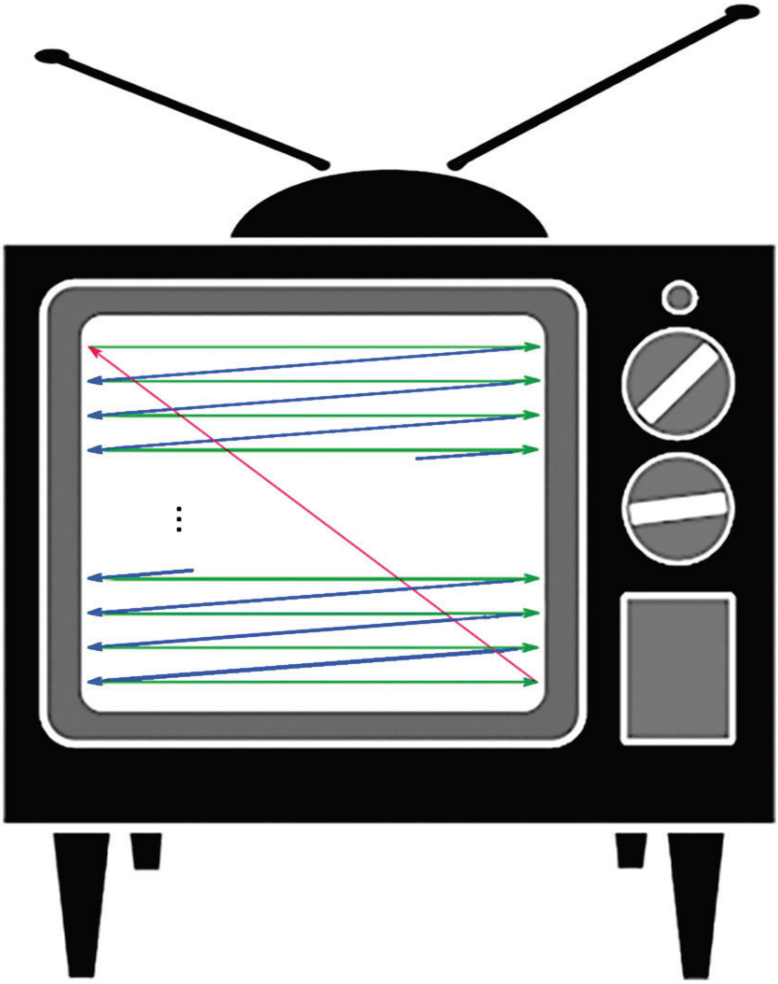
ואולם, עדיין לא עסקנו בשאלה מה עושה אותו "אקדח אלקטרונים" לאחר שסיים את מסעו והגיע לקצהו הימני התחתון של המסך.

ובכן, השכל הישר מלמד אותנו כי על "אקדח האלקטרונים" לבצע עתה דילוג אלכסוני בחזרה אל הנקודה השמאלית העליונה של המסך, ולהמתין שם עד לפעם הבאה אשר יידרש להתחיל מחדש במשימת ציור המסך.

האיור הבא ממחיש כיצד מתנועע "אקדח האלקטרונים" על פני מסך הטלוויזיה:
הקווים הירוקים ממחישים כיצד מציירת קרן האלקטרונים את המסך שורה אחר שורה משמאל לימין, ומלמעלה למטה.

הקווים הכחולים מייצגים את זמן ההפוגה בין סריקת השורה האחת ועד השורה העוקבת, דהיינו את פרק הזמן המשמש ל-DLI.

הקו האדום מייצג את זמן ההפוגה בין סיום מחזור אחד של סריקת המסך כולו ועד לתחילת סריקת המסך הבאה, דהיינו פרק הזמן המשמש ל-VBI.



? כמה זמן נותר ״אקדח האלקטרונים״ כבוי משעה שהגיע לנקודה הימנית התחתונה ועד שהוא מתחיל מחדש את מסעו מן הנקודה השמאלית העליונה?

כמה חלקיקי שנייה – או ליתר דיוק, 20,000 מחזורים!

פרק זמן זה עצום במונחי המחשב, והוא מותיר למתכנת מקום לכתוב רוטינות ארוכות למדיי שתפעלנה ברקע מבלי להפריע לפעולה התקינה של סריקת המסך.

ניצולה של אותו מרווח בין סיום מחזור אחד של ציור מסך ובין תחילת המחזור הבא על מנת לבצע משימות, נקרא: "פסיקת המרווח האנכי", או **באנגלית:** Vertical Blank Interrupt **, ובקיצור:** VBI.

דע לך כי זמן המרווח האנכי משמש לא רק אותך המתכנת, אלא גם את מערכת ההפעלה של המחשב. גם היא עושה שימוש נרחב בפרק הזמן הזה עבור משימות המחשב השוטפות, משימות כגון עדכון השעון הפנימי של המחשב (השמור כזכור בבתים 18-20); סנכרון בין ערכי הרגיסטרים של הצבע בכתובות הזיכרון הגבוהות לבין ערכיהם בזיכרון ה-Shadow ועוד. לצורך ביצוע שלל משימות אלו, שמורה בתוך זיכרון ה-ROM רוטינה ייחודית (הכתובה בשפת מכונה כמובן!) – רוטינה אשר מתבצעת שוב ושוב, בכל הפוגה אנכית, באמצעות הפרעת VBI.

? עכשיו נשאלת השאלה כיצד נוכל להוסיף בפרק הזמן המיוחד הזה פקודות משלנו?

דרך אחת היא פשוט לכתוב רוטינה **חליפית** – רוטינה אשר תכלול את שלל הפקודות של מערכת ההפעלה ובנוסף להן כמה פקודות משלנו. רוב המתכנתים לא ירצו להשתמש בדרך הזו, משום שהיא מסורבלת לביצוע.

לחלופין אפשר לכתוב רוטינה ייחודית משלנו שתכלול אך רק את הפקודות הרצויות לנו. רוטינה אשר תפעל כהפרעת VBI לכל דבר, אך עם סיומה תדאג לשלוח את המחשב להפרעת VBI **נוספת** – אשר במהלכה תבוצע הרוטינה המקורית של מערכת ההפעלה.

למזלנו, מהנדסי מחשב האטארי התחשבו בצורך התכנותי הזה ואפשרו לנו, המתכנתים, להשתמש בדרך השנייה, כלומר להפעיל בזו אחר זו שתי רוטינות VBI: הראשונה רוטינה משלנו, והשנייה רוטינה של מערכת ההפעלה. לצורך כך, הפרעת VBI תוכננה מלכתחילה כ**הפרעה דו-שלבית**.

כל אימת שאנו מאתחלים את המחשב, נרשמות שתי כתובות זיכרון במקום מסוים ב-RAM. האחת נקראת VBI "מיידי", והשנייה –VBI "דחוי". אנו יכולים כמובן להחליף כתובות אלו בכתובות אחרות, כרצוננו.

בכל אימת ש"אקדח האלקטרונים" מגיע לנקודה הימנית התחתונה של המסך, המעבד המרכזי נשלח אל הכתובת הראשונה מבין השתיים, של ה-VBI המיידי. אם אכן קיימת שם רוטינה, הוא ממהר לבצעה. מייד אחר כך, המעבד המרכזי נשלח לכתובת הזיכרון השנייה, של ה-VBI הדחוי, וכתובת זו שולחת אותו לרוטינה המקורית של מערכת ההפעלה.

? אז איך עושים את זה?

בניית VBI היא משימה פשוטה מאוד.

כל שעלינו לעשות הוא:

1. להגדיר את כתובת הזיכרון הראשונה (המפנה ל-VBI המיידי) אשר שמורה בזוג הבתים המצביעים 549, 548.

2. לבנות את הרוטינה, כשאנו דואגים שהיציאה להפרעה תהייה מבלי להותיר עקבות.

3. עם סיום הרוטינה להפנות את המעבד המרכזי, באמצעות פקודת האסמבלי **JMP** אל הרוטינה המקורית של מערכת ההפעלה (היושבת כמובן ב-ROM).

נפתח במשימה פשוטה: הפיכת הסמן (cursor) למהבהב.

בפרק י"א במחשבת 8 בנינו רוטינה שגורמת לסמן להבהב. עתה ניצור רוטינה דומה שעושה זאת כהפרעת VBI.

ראשית, נבנה את החלק המרכזי של הרוטינה הנוגע להתנהגות הסמן. על מנת שלא להתבלבל בין רוטינות VBI ורוטינות DLI, נעשה לנו מנהג לכתוב את רוטינות ה-VBI באמצע דף 6, כלומר בכתובת $650.

```
10     *=$650
20  TEMP
40     LDA  $14
50     AND  #$10
60     LSR  A
70     LSR  A
80     LSR  A
90     STA  $2F3
110     JMP  TEMP
```

שים לב, כדי לבחון את פעולת הרוטינה, הוספתי באופן זמני לולאה אינסופית.

אם אינך מבין את שורות הקוד היוצרות את הבהוב הסמן, עיין בפרקים י-י"א במחשבת 8.

עתה עלינו להוסיף את פקודת הדילוג **JMP** אל רוטינת ה-VBI המקורית של מערכת ההפעלה. שים לב, במסגרת יחידת לימוד זו, לא אכנס להסבר מפורט כיצד בדיוק מתבצע אותו מעבר מרוטינת ה-VBI שלנו אל הרוטינה של מערכת ההפעלה. אציין רק שכדי שאותו מעבר יתרחש בצורה חלקה, עלינו להפנות את המעבד המרכזי אל כתובת הזיכרון: **58466** (או $E462).

נוסיף אם כך את שורת הקוד:

```
120    JMP  $E462
```

לבסוף, עלינו לדאוג שרוטינת ההפרעה שלנו תתבצע מבלי להותיר עקבות.

שים לב – כאשר כתבנו רוטינות DLI, דאגנו לשמור את ערכי הרגיסטרים (A, X, Y) אשר עשינו בהם שימוש, ולהחזירם לקדמותם עם סיום הרוטינה. כאשר אנו עוסקים ברוטינות VBI, עלינו לדאוג – **בנוסף לכך** – שדגלוני המעבד המרכזי גם הם ישובו לקדמותם עם סיום הרוטינה.

לשם כך נשתמש בזוג הפקודות **PHP** **PLP** (אם אינך זוכר את מהות פעולתן, רענן את זיכרונך בפרק ט' במחשבת 8).

נוסיף אפוא את הפקודות (ואגב כך, נדרוס את הלולאה הזמנית שיצרנו).

```
20     PHP
30     PHA
100     PLA
110     PLP
```

רוטינת ה-VBI שלנו מוכנה לשימוש!

כל שנותר לנו לעשות הוא להפוך את תוכנת האסמבלי לקוד בשפת מכונה וכהרגלנו, להפעיל את הקוד הזה מתוך תוכנת בייסיק מתאימה.

כך עשויה אפוא להיראות תוכנת הבייסיק שלנו:

```
10 FOR B=1664 TO 1680:READ K:POKE B,K:
NEXT B
20 POKE 548,128:POKE 549,6
30 DATA 8,72,165, 20,41,16,74,74,74,
141,243,2,104,40,76,98,228
```

מוסיפים מוסיקת רקע

עיקרון המחזוריות של ה-VBI (כמו גם של ה-DLI), יכול לשמש אותנו ליצירת לולאות מותנות או אינסופיות, שתשמשנה אותנו לצרכים תכנותיים מגוונים. אחת האפשרויות היא להוסיף מוסיקת רקע שתנוגן כל העת ללא הפסקה וללא תלות בתוכנה שפועלת במקביל (כפי שבנויים משחקי מחשב רבים).

אך ראשית, כמה מילים על שימוש בצלילי המחשב.

עד כה, בעבודה בסביבת בייסיק, יצרנו צלילים באמצעות הפקודה: **SOUND**.
על מנת לנגן את הצליל "דו" לדוגמה, פקדנו על המחשב:

```
SOUND 0,121,10,8
```

אתה ודאי זוכר מה משמעות כל אחד מארבעת הפרמטרים המלווים את הפקודה (אם לא, עיין בפרק "התפוצצויות בחלל" במחשבת 3).

דע לך כי אפשר לייצר צלילים גם באמצעות פניות ישירות לזיכרון.

זוג הבתים: 53760, 53761 (או: $D200, $D201) – מטפל בזמר מס' 0.

זוג הבתים: 53762, 53763 (או: $D202, $D203) – מטפל בזמר מס' 1.

זוג הבתים: 53764, 53765 (או: $D204, $D205) – מטפל בזמר מס' 2.

וזוג הבתים: 53766, 53767 (או: $D206, $D207) – מטפל בזמר מס' 3.

כל אחד מארבעת הבתים הזוגיים (53760, 53762, 53764, 53766) אחראי **לטון** של הצליל. כך למשל, על מנת להפיק את הצליל "דו" בדוגמה לעיל, נפקוד לתוך הבית 53760 את הערך 121 (או: $79).

באשר לפרמטר ה"טוהר" ופרמטר העוצמה של הצליל – אל שני הפרמטרים הללו (המאוגדים יחדיו כזוג) אחראי כל אחד מארבעת הבתים האי-זוגיים (53761, 53763, 53765, 53767). על מנת לחשב את הערך שיש לפקוד לתוך בתים אלו, נכפיל את ערך פרמטר ה"טוהר" ב-16 ונוסיף לתוצאה את ערך פרמטר העוצמה.

אם נחזור לדוגמה לעיל, על מנת להפיק את הצליל האמור, יש לפקוד לתוך בית 53761 את הערך: 168 (או: $A8) (שהוא תוצאת התרגיל: 16X10+8).

עתה, נפנה לכתיבת רוטינת VBI שתנגן ברקע קטע קצר של מוסיקה – את אותם 6 צלילים מוכרים מה-Self-Test !

לצורך המשימה ניעזר בשעון השניות של המחשב, והוא ישמש לנו מעין מטרונום. מכיוון ששניים מבין ששת הצלילים קצרים, וארבעת האחרים

ארוכים פי 2, נצטרך בסך הכול 10 יחידות צליל. בנוסף, נתווה 6 יחידות שקט (שיפריד בין המנגינות) – ובסך הכול 16 פעימות של "מטרונום".

על מנת לפשט את העניינים, נוכל לקבוע כי ניגון מקטע אחד של המנגינה (כולל ההפוגה השקטה) יארך בדיוק "דקת מחשב" אחת. אנו יודעים כבר כי ב"דקת מחשב" יש בסך הכול 256 "שניות מחשב", ולכן, בעזרת כמה פקודות לוגיות פשוטות נוכל להפוך את מניית שניות המחשב למנייה של 16 פעימות.

- כתוב רוטינת VBI שתשלוף בכל יציאה להפרעה, את הצליל הנכון ותנגנו. לנוחיותך, אלו הם ערכי התווים של המנגינה: $51, $5B, $44, $3C, $2D, $35 הפוך את הרוטינה לקוד בשפת מכונה והטמע אותו בתוכנת בייסיק.

[תשובה 20 בעמ' 114]

אנימציה

בפרק 10 במחשבת 8 כתבנו תוכנה המייצרת אנימציה תוך שימוש באחד מן הסימנים הגרפיים. עתה, לאחר שצברנו ידע ב-VBI, נוכל לייצר אנימציה שתפעל כל העת ברקע מבלי להפריע לנו במשימות תכנותיות אחרות.

- תרגל את עצמך (!) וכתוב תוכנת אסמבלי אשר מעתיקה את מערך הסימנים ממקומו ב-ROM אל מקום חדש בזיכרון ה-RAM.

הוסף לתו "a" את הסימן ה"אקצנט" העליון אשר ישמש בהמשך לבניית האנימציה.

אל תשכח להוסיף את הפקודות הנדרשות להרצת הרוטינה מתוך תוכנת בייסיק.

[תשובה 21 בעמ' 116]

■ הפוך את התוכנה לקוד שפת מכונה שאותו נריץ בהמשך מתוך תוכנית בייסיק באמצעות הפקודה **USR**

כעת נכתוב את רוטינת ה-VBI אשר תיצור אנימציה מתוך התו "ā".

ראשית, ניעזר שוב בשניות המחשב וניצור באמצעותן "מטרונום" פשוט אשר יפיק כשש "פעימות" בשנייה. כל עוד אין "פעימה", נפקוד על המחשב לסיים מיד את רוטינת ה-VBI.

```
10      *=$650
40      LDA  $14
50      AND  #$7
60      CMP  #$7
70      BNE  END
```

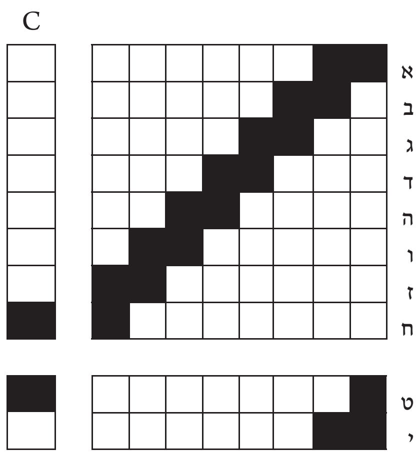
עתה נוסיף את הפקודות המייצרות את האנימציה.

שים לב, בשונה מהתוכנית שכתבנו במחשבת 8, הפעם עלינו לקחת בחשבון שני עניינים:

ראשית, בכל יציאה להפרעת VBI, יש לקרוא מחדש את המידע השמור בבית הראשון של נתוני התו "a", לעדכן אותו, ולקדם על סמך המידע המעודכן גם את שבעת הבתים העוקבים (המייצרים יחדיו את התו).

שנית, מכיוון שהפקודה **ROL** נעזרת כזכור בדגל C, חייבים להדליק את הדגל באופן יזום בשני מצבי הקיצון – הימני ביותר והשמאלי ביותר. אם לא נעשה כן, דגל C "ישכח" את המידע השמור בו והאנימציה תלך ותעלם...

להלן סקיצה שתעזור להבין את סדר הפעולות הנדרש:



נקל לראות כי כאשר הבית הראשון של נתוני התו "a", נמצא באחד משני המצבים – ח ו-ט, חייבים להדליק באופן יזום את הדגל C.

נכתוב, אם כן את שורות הקוד המתאימות:

```
80    LDA $9B08
90    CMP #$1
100    BEQ SET
110    CMP #$80
120    BEQ SET
130    CLC
140  CONT
150    ROL A
160    STA $9B08
170    ROL A
180    STA $9B09
190    ROL A
200    STA $9B0A
210    ROL A
220    STA $9B0B
230    ROL A
240    STA $9B0C
250    ROL A
260    STA $9B0D
270    ROL A
280    STA $9B0E
290    ROL A
300    STA $9B0F
310  END
340    JMP $E462
350  SET
360    SEC
370    JMP CONT
```

נוסיף את שורות הקוד המבטיחות יציאה להפרעת VBI "מבלי להותיר עקבות":

```
20      PHP
30      PHA
320      PLA
330      PLP
```

■ הפוך את הרוטינה לקוד שפת מכונה והוסף אותו לתוכנית הבייסיק שלך, כך שתפעל כהפרעת VBI.

[תשובה 22 בעמ' 117]

סיכום

בפרק זה סקרנו את פסיקת (הפרעת) המרווח האנכי (Vertical Blank Interrupt). פסיקה זו חוזרת על עצמה 50 פעמים בשנייה והיא מנוצלת על ידי מערכת ההפעלה של מחשב האטארי לביצוע משימות שוטפות. למדנו לנצל את אופייה הדו-שלבי על מנת להוסיף לה הוראות משלנו שתבוצענה באופן מחזורי, מבלי להפריע לפעולות השוטפות של המחשב.

ניתן להכליל ולומר כי אם עלינו לבצע שינויים בפרמטרים על **אזורים מסוימים** במסך, נשתמש בפסיקות ה-DLI, ואם ברצוננו לבצע פעולות מחזוריות (הן על פרמטרים של המסך והן בכלל), נשתמש בפסיקות ה-VBI.

כאן אציין כי ישנן גם דרכים נוספות ליצור לולאות מותנות או אינסופיות שתפעלנה ברקע באופן מחזורי מבלי להשתמש בפסיקות ה-VBI או ה-DLI. על כך יפורט בחוברות הבאות.

נספח 1
הרחבה נוספת (למתקדמים)
על פקודות ה-Antic

א. סקירה כללית על ה-Antic

ה-Antic, כפי שאנו כבר יודעים, הוא מיקרו-מעבד (שבב) המטפל בסריקת (ציור) המסך של מחשב האטארי והוא מתואם עם עבודת המעבד המרכזי, הלוא הוא ה-6502.

את ה-Antic אפשר לדמות לנהג של המעבד המרכזי, וזאת משום שהוא יכול לעצור את המעבד המרכזי בעת הצורך. כאן עליך להבין כי הארכיטקטורה של מחשב האטארי מאפשרת בכל רגע נתון גישה אל הזיכרון הישיר (Direct Memory Access) אך ורק לשבב אחד, ולכן כאשר ה-Antic נדרש לקרוא מן הזיכרון, הוא חייב לעצור לרגע את המעבד המרכזי. משך הזמן אשר אותו "גונב" ה-Antic תלוי בגורמים רבים, כגון: מהו המצב הגרפי הפעיל, והאם גרפיקת השחקנים נמצאת בשימוש.

ה-Antic יודע לקרוא את פקודות ה-Display List ("מתווה התצוגה"). כל פקודה שכזו מורה לו לצייר מספר מסוים של קווי סריקה ובדרך מסוימת. קווי הסריקה הללו מצוירים על המסך ללא צורך בהתערבות נוספת של המעבד המרכזי, אם כי אפשר כמובן לפקוד על המעבד המרכזי כן להתערב, וזאת באמצעות רוטינות DLI.

כל Display List סטנדרטי פותח ב-24 שורות ריקות.[7] מיד אחר כך נמתחות 192 שורות סריקה של נתוני תצוגה. הפוגת המרווח האנכי תתרחש תמיד לאחר סריקת כל השורות הממלאות את המסך (לרבות השורות הריקות).

ב. פקודת ה-Antic והלוגיקה שמאחוריהן

כאשר ה-Antic קורא את מתווה התצוגה, הוא פוגש למעשה פקודות תצוגה ייחודיות לו (בשונה למשל מפקודות שפת המכונה שהכרנו במחשבת 8). מבנה הפקודות של ה-Antic מורכב מבית אחד בלבד או מבית אחד בתוספת כתובת של זוג בתים מצביעים. נבחין בין 4 סוגים של פקודות:

(1) שורות ריקות
(2) יחידות גרפיקה מסוג טקסט
(3) יחידות גרפיקה מסוג Bitmap ("מפת סיביות")
(4) הוראות קפיצה

הפקודות מקודדות אל תוך הבית באמצעות 8 הסיביות: ארבעת הסיביות הנמוכות מקודדות את **סוג הפקודה** ואילו ארבעת הסיביות הגבוהות מקודדות מעין "דגלונים" המשפיעים על פקודה זו:

נפתח בארבעת הסיביות הנמוכות המקודדות כאמור את סוג הפקודה.

באמצעות ארבע סיביות אלו ניתן לייצג את כל 14 היחידות הגרפיות הקיימות באטארי. שים לב, חלק מהיחידות הגרפיות מוכרות לך מן המסכים הגרפיים המובנים בבייסיק, אך לא כולן!

7 דע לך כי עוד לפני 24 השורות הריקות המוגדרות ב-Display List, נמתחות באופן אוטומטי 8 שורות ריקות בראש מסך הטלוויזיה).

במחשב האטארי ישנן 6 יחידות גרפיות של טקסט:

קוד ה-Antic של היחידה הגרפית (בשיטה העשרונית)	מסך מקביל בבייסיק	שורות X עמודות	שורות סריקה (לכל יחידה)	מספר צבעים
2	Gr. 0	40 x 24	8	2
3	–	40 x 19	10	2
4	–	40 x 24	8	4
5	–	40 x 12	16	4
6	Gr. 1	20 x 24	8	5
7	Gr. 2	20 x 12	16	5

בנוסף ישנן 8 יחידות גרפיות של מסכי "מפת סיביות".

קוד ה-Antic של היחידה הגרפית (בשיטה העשרונית)	מסך מקביל בבייסיק	שורות X עמודות	שורות סריקה (לכל יחידה)	מספר צבעים
8	Gr. 3	40 x 24	8	4
9	Gr. 4	80 x 48	4	2
10	Gr. 5	80 x 48	4	4
11	Gr. 6	160 x 96	2	2
12	Gr. 14	160 x 192	1	2
13	Gr. 7	160 x 96	2	4
14	Gr. 15	160 x 192	1	4
15	Gr. 8	320 x 192	1	2*

ארבעת הסיביות הגבוהות משמשות מעין "דגלונים":

- הדלקת הסיבית החמישית (D4) מאפשרת גלילה אופקית של היחידה הגרפית.
- הדלקת הסיבית השישית (D5) מאפשרת גלילה אנכית של היחידה הגרפית.
- הדלקת הסיבית השביעית (D6) מורה ל-Antic כי בנוסף לסריקה האופקית, עליו לגשת אל הזיכרון ולאתר את נתוני המסך. לשם כך נדרשת כתובת של זוג בתים מצביעים.
- הדלקת הסיבית השמינית (D7) מאפשרת DLI, כלומר הפרעה (פסיקה) של מתווה התצוגה.

? ומה באשר לשורות ריקות?

ובכן שורות ריקות מקבלות ערך 0 בארבעת הסיביות הנמוכות ואילו שלושת הסיביות הגבוהות יותר (החמישית, השישית והשביעית) משמשות לציין את כמות השורות הריקות.

כך, הערך 0 ימתח שורה ריקה אחת; הערך 16 ימתח 2 שורות ריקות; הערך 32 ימתח 3 שורות ריקות; הערך 48 ימתח 4 שורות ריקות; הערך 64 ימתח 5 שורות ריקות; הערך 80 ימתח 6 שורות ריקות; הערך 96 ימתח 7 שורות ריקות; והערך 112 ימתח 8 שורות ריקות.

שים לב, הדלקת הסיבית השמינית מאפשרת DLI גם עבור שורה/שורות ריקות!

? ומה לגבי הערך 1?

ובכן, הדלקת הסיבית הראשונה היא למעשה פקודת קפיצה. על מנת להודיע ל-Antic להיכן עליו לקפוץ, עלינו להדליק כמובן את הסיבית השביעית (שכזכור מורה ל-Antic כי עליו לגשת את מקום מסוים בזיכרון, והיא מצפה לזוג בתים מצביעים).

אשר על כן, הדלקת הסיבית הראשונה והשביעית נותנים את הערך 65, שאותו אנחנו מכירים כמובן ממתווי התצוגה שראינו בפרקים הקודמים.

ג. הדגמה קצרה

בפרק הראשון לעיל עיינו במתווה התצוגה (Display List) של מסך גרפי מס' 3.

אציג את ה-Display List שוב, ולצורך הדיון אצבע באדום אך ורק את פקודות ה-Antic:

112,112,112,72,112,158,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,66,96,159,2,2,2,
65,78,158

כעת אנתח כל אחת מן הפקודות, ולשם כך אציג אותן בשיטה הבינארית:

הפקודה (עשרונית)	הפקודה (בינארית)		הסבר
(112)	0000	0111	ארבעת הסיביות הנמוכות כבויות, ולכן נמתחת שורה ריקה. ערכן של הסיביות הגבוהות הוא 7 ולכן נמתחות עוד 7 שורות ריקות, ובסך הכול – 8.
(...)+(72)	1000	0100	ערכן של ארבעת הסיביות הנמוכות הוא 8, ולכן נמתחת יחידה גרפית מסוג גרפיקה 3. באזור הסיביות הגבוהות, דלוקה הסיבית השביעית (D6), ולכן ה-Antic מצפה לזוג בתים מצביעים שיורו לו היכן יושב זיכרון המסך.
(8)	1000	0000	ערכן של ארבעת הסיביות הנמוכות הוא 8, ולכן נמתחת יחידה גרפית מסוג גרפיקה 3.
(...)+(66)	0010	0100	ערכן של ארבעת הסיביות הנמוכות הוא 2, ולכן נמתחת יחידה גרפית מסוג גרפיקה 0. באזור הסיביות הגבוהות, דלוקה הסיבית השביעית (D6), ולכן ה-Antic מצפה לזוג בתים מצביעים שיורו לו היכן יושב זיכרון המסך.
(2)	0010	0000	ערכן של ארבעת הסיביות הנמוכות הוא 2, ולכן נמתחת יחידה גרפית מסוג גרפיקה 0.
(...)+(65)	0001	0100	ערכן של ארבעת הסיביות הנמוכות הוא 1, ולכן ה-Antic יודע שעליו לקפוץ. באזור הסיביות הגבוהות, דלוקה הסיבית השביעית (D6), ולכן ה-Antic מצפה לזוג בתים מצביעים שיורו לו להיכן עליו לקפוץ.

• אם ברצוננו להכניס פקודת הפרעה (פסיקה) באחת השורות, נדליק עבורה את הסיבית השמינית (D7):

הפקודה (בינארית)		הפקודה (עשרונית)
1000	1000	(136)

• אם ברצוננו לאפשר גלילה אופקית עדינה באחת השורות, נדליק עבורה את הסיבית החמישית (D4):

הפקודה (בינארית)		הפקודה (עשרונית)
0001	1000	(24)

• אם ברצוננו לאפשר גלילה אנכית עדינה באחת השורות, נדליק עבורה את הסיבית השישית (D5):

הפקודה (בינארית)		הפקודה (עשרונית)
0010	1000	(40)

נספח 2
מדריך להפיכת קוד אסמבלי לקוד שפת מכונה בסביבת בייסיק

במחשבת 8 למדת להפוך קוד אסמבלי לקוד שפת מכונה באמצעות כתבן האסמבלר. לצורך כך, משמשת פקודת הכתבן ASM.

פקודה זו כזכור בודקת את תקינות הקוד שכתבת, ממירה אותו לקוד שפת מכונה וכותבת אותו במקום המתאים בזיכרון, כפי שהגדרת.

על מנת להזין את אותו קוד לתוכנת הבייסיק (אל תוך הטווח השמור תחת פקודת הבייסיק DATA) נוכל לבנות תוכנה עזר קצרה בבייסיק אשר תקרא מתוך הדיסקט את הקוד ותציג אותו על המסך (בשיטה העשרונית כמובן!).

ניקח לדוגמה קוד קצר באסמבלי, הצובע את המסך באדום:

```
10    *=$600
20    LDA #$22
30    STA $2C6
```

נפקוד על המחשב ASM ונשמור את התוכנה באמצעות הפקודה SAVE:

```
SAVE #D:CODE1<600,609
```

שים לב, כשאתה שומר בדיסקט את הקוד בשפת מכונה, הקפד להוסיף טווח ביטחון של לפחות 6 בתים קדימה – בהמשך תבין מדוע.

בשלב זה נעבור לסביבת הבייסיק ונכתוב תוכנה קצרה שתקרא את נתוני קוד שפת המכונה:

```
10 OPEN #1,4,0,"D:CODE1.TXT"
20 TRAP 50
30 GET #1,A:?A;",";
40 GOTO 30
50 END
```

כשנריץ את התוכנה, נקבל את המסך את המספרים האלו:

```
255,255,0,6,9,6,169,34,141,198,2,0,0,0,0,0
```

הערכים הצבועים באדום (החל מן הבית השביעי ואילך), הם המספרים המעניינים אותנו ואותם יש להקליד אל תוך תוכנת הבייסיק.

? ולמה משמשים 6 הערכים הראשונים?

ובכן, ערכים אלו מורים למחשב – בזמן קריאת התוכנה מן הדיסקט – היכן בזיכרון יש לטעון את המידע. כאן עליך לזכור כי את הקובץ השמור על הדיסקט תוכל לטעון אל כתבן האסמבלר באמצעות פקודת הכתבן **LOAD** ולצורך כך, על הכתבן לדעת היכן בזיכרון לטעון את המידע.

נתבונן על הערכים השלישי, הרביעי החמישי והשישי. השניים הראשונים: 0, 6 מציינים (בשיטת ה-MSD, LSD) את כתובת הזיכרון הראשונה שבה אמור להישמר הקוד, ואילו שני הערכים העוקבים: 9, 6 מציינים את כתובת הזיכרון האחרונה שבה אמור להישמר הקוד.

תשובות

[תשובה 1 מעמוד 10]

```
10 GR. 0
20 DLIST=PEEK(561)*256+PEEK(560)
30 FOR I=0 TO 31
40 ? PEEK(DLIST+I);",";
50 NEXT I
```

[תשובה 2 מעמוד 16]

```
10 GRAPHICS 3
20 DLIST=PEEK(561)*256+PEEK(560)
30 FOR I=0 TO 33
40 READ M: POKE DLIST+I, M
50 NEXT I
60 DATA 112,112,112,66,96,159,2,2,2,72,
112,158,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
8,8,8,8,8,65,78,158
```

[תשובה 3 מעמוד 22]

```
10 GRAPHICS 6: POKE 559,0
20 DLIST=PEEK(561)*256+PEEK(560)
30 FOR I=0 TO 71: READ R: POKE DLIST+I,R:
NEXT I
```

```
40 POKE 559,34
100 DATA 112,112,112,70,224,151,6,6,6
110 DATA 72,48,152,8,8,8,8,8,8,8,75,
128,152,11,11,11,11,11,11,11,11,11, 11,
11,11,11,11,11,11,11
120 DATA 11,11,11,11,11,11,11,11,11,11,
11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
11,11,11,11,11,11,11,11,65,130,151
```

[תשובה 4 מעמ' 24]

```
10 DIM A(200)
20 GRAPHICS 8
30 DLIST=PEEK(561)*256+PEEK(560)
40 FOR I=0 TO 199:A(I+1)=PEEK(DLIST+I):
NEXT I
50 GRAPHICS 0
60 FOR I=1 TO 200:? A(I);",";:NEXT I
```

[תשובה 5 מעמ' 29]

```
10 DLIST=PEEK(561)*256+PEEK(560)
20 L=DLIST+4:M=DLIST+5
30 POKE L,PEEK(L)+1
```

[תשובה 6 מעמ' 29]

```
10 GRAPHICS 0
20 DLIST=PEEK(561)*256+PEEK (560)
30 ?: ? "THAT'S ONE SMALL STEP FOR A MAN"
40 FOR X=PEEK(DLIST+4) TO 40+PEEK(DLIST+4)
50 POKE DLIST+4,X
60 FOR TIME=1 TO 100:NEXT TIME
70 NEXT X
```

[תשובה 7 מעמ' 34]

```
10 GRAPHICS 0:POKE 82,0
20 DLIST=PEEK(561)*256+PEEK(560)
30 FOR C=0 TO 31:READ R:POKE DLIST+C,R:
NEXT C
40 ? "-WELCOME TO THE MOON-"
50 ?: ? "THAT'S ONE SMALL STEP FOR A
MAN, ONE GIANT LEAP FOR MANKIND..."
60 ?: ?: ? "(NIEL ARMSTRONG)"
70 FOR L=1 TO 4
80 FOR M=104 TO 210: POKE DLIST+8,M:
FOR TIME=1 TO 200: NEXT TIME: NEXT M
90 NEXT L
100 DATA 112,112,112,66,64,156,2,66,
104,156,66,8,157,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,65,32,156
```

[תשובה 8 מעמ' 34]

```
20 GRAPHICS 3:DEG:COLOR 1
30 FOR X=0 TO 360 STEP 6
40 PLOT 30+9*SIN(X),9+9*COS(X)
50 NEXT X
```

[תשובה 9 מעמ' 54]

```
10    *=$682
20     PLA
30     PLA
40     PLA
50     LDX $14
60     INX
70 LOOP
80     CPX $14
90      BEQ END
100     JMP LOOP
110 END
120     STA $D404
130     RTS
```

[תשובה 10 מעמ' 54]

```
180 IF D=-1 THEN 220
220 M=0:RESTORE 950
230 READ D
235 IF D=-1 THEN 260
240 POKE 1666+M,D
250 M=M+1
255 GOTO 230
610 U=USR(1666,15):U2=USR(1626)
615 U=USR(1666,FINE)
```

```
950 DATA 104,104,104,166,20,232,228,
20,240,3,76,133,6,141,4,212,96,-1
```

[תשובה 11 מעמ' 57]

```
10 DIM WORD$(8)
110 GRAPHICS 0:SETCOLOR 2,0,0
120 DLIST=PEEK(561)*256+PEEK(560):
SCR=PEEK(89)*256+PEEEK(88)
130 POKE DLIST+3,71+32
140 FOR C=6 TO 17:POKE DLIST+C,7+32:
NEXT C
150 POKE DLIST+18,112:POKE DLIST+19,112
160 FOR C=0 TO 19
170 READ WORD$
180 FOR L=1 TO LEN(WORD$)
190 POKE SCR+(20*C)+L,ASC(WORD$(L))-32
200 NEXT L:NEXT C
210 FOR LOOP=1 TO 20
220 FOR F=1 TO 15
223 POKE 54277,F
226 FOR TIME=1 TO 100:NEXT TIME
230 NEXT F
240 TMP=PEEK(DLIST+5)*256+PEEK(DLIST+4)
+20
243 T1=INT(TMP/256):T2=TMP-(T1*256)
246 POKE DLIST+5,T1:POKE DLIST+4,T2
250 NEXT LOOP
900 DATA ONE,TWO,THREE,FOUR,FIVE,SIX,
SEVEN,EIGHT,NINE,TEN,ELEVEN,TWELVE,
THIRTEEN,FOURTEEN,FIFTEEN,SIXTEEN
910 DATA SEVENTEEN,EIGHTEEN,NINETEEN,
TWENTY
```

[תשובה 12 מעמ' 58]

```
10     *=$650
20  DLST=$9C20
30     PLA
40     LDA #$0
50     LDY #$4
70     LDX $14
80     INX
90  LOOP
100     CPX $14
110     BEQ END
120     JMP LOOP
130  END
140     STA $D405
150     LDA DLST,Y
160     CLC
170     ADC #$14
180     STA DLST,Y
190     INY
200     LDA DLST,Y
210     ADC #$0
220     STA DLST,Y
230     RTS
```

[תשובה 13 מעמ' 58]

אלו השורות שיש להוסיף או להחליף (לדרוס):

```
20 READ D
30 IF D=-1 THEN 60
40 POKE 1536+C,D
```

```
50 C=C+1:GOTO 20
60 C=0:RESTORE 600
70 READ D
80 IF D=-1 THEN 110
90 POKE 1616+C,D
100 C=C+1:GOTO 70
223 U=USR(1536,F)
226 <למחוק את השורה>
240 U=USR(1616)
243 <למחוק את השורה>
246 <למחוק את השורה>
500 DATA 104,104,104,166,20,232,228,
20,240,3,76,6,6,141,5,212,96,-1
600 DATA 104,169,0,160,4,166,20,232,
228,20,240,3,76,88,6,141,5,212,185,
32,156,24
610 DATA 105,20,153,32,156,200,185,
32,156,105,0,153,32,156,96,-1
```

[תשובה 14 מעמ' 73]

```
10    *=$600
20    PHA
30    TXA
40    PHA
50    LDX #$B1
60 LOOP
```

```
70     STX  $D40A
80     STX  $D018
90     DEX
100     BNE  LOOP
110     PLA
120     TAX
130     PLA
140     RTI
```

(תשובה 15 מעמ' 74]

```
10     *=$600
20     PHA
30     TXA
40     PHA
50     TYA
60     PHA
70     LDX  $14
80     LDY  #$B1
90  LOOP
100     STX  $D40A
110     STX  $D018
120     DEX
130     DEY
140     BNE  LOOP
150     STY  $D018
160     PLA
```

```
170     TAY
180     PLA
190     TAX
200     PLA
210     RTI
```

[תשובה 16 מעמ' 76]

```
10 DLIST=PEEK(561)*256+PEEK(560)
20 FOR C=6 TO 28:POKE DLIST+C,130:NEXT C
30 READ A
40 IF A=-1 THEN 70
50 POKE 1536+M,A
60 M=M+1:GOTO 30
70 POKE 512,0:POKE 513,6
80 POKE 54286,192
100  DATA 72,173,11,212,141,10,212,141,
24,208,104,64,-1
```

[תשובה 17 מעמ' 76]

```
34     AND #$4
37     EOR #$94
```

ובבייסיק:

```
100  DATA 72,173,11,212,41,4,73,148,141,
10,212,141,24,208,104,64,-1
```

[תשובה 18 מעמ' 79]

```
10 DLIST=PEEK(561)*256+PEEK(560)
20 POKE DLIST+6,2+128
30 POKE DLIST+14,2+128
40 POKE DLIST+21,2+128
50 READ A:IF A=-1 THEN 80
60 POKE 1536+I,A
70 I=I+1:GOTO 50
80 POKE 512,0:POKE 513,6
90 POKE 54286,192
100 DATA 72,138,72, 174,40,6,189,37,6,
141,10,212,141,24,208,238,40,6,174,40,
6,224,3,240,4,104,170,104,64
110 DATA 162,0,142,40,6,76,25,6,68,254,
198,-1
```

[תשובה 19 מעמ' 81]

```
10    *=$600
20    PHA
30    LDA #$90
40    STA $D409
50    PLA
60    RTI
```

[תשובה 20 מעמ' 91]

```
10    *=$650
20    PHP
30    PHA
40    TXA
50    PHA
60    LDA $14
70    AND #$F0
80    LSR A
90    LSR A
100    LSR A
110    LSR A
120    TAX
130    LDA DATA,X
140    STA $D200
150    LDA #$AA
160    STA $D201
170    PLA
180    TAX
190    PLA
200    PLP
210    JMP $E462
230 DATA
240    .BYTE $51, $51, $5B, $5B, $44,
$44, $3C, $2D, $35, $35
250    .BYTE $00, $00, $00, $00
```

[תשובה 21 מעמ' 92]

```
10     *=$600
20     PLA
30     LDX #$0
40 LOOP INX
50     LDA $E000,X
60     STA $9800,X
70     LDA $E100,X
80     STA $9900,X
90     LDA $E200,X
100     STA $9A00,X
110     LDA $E300,X
120     STA $9B00,X
130     CPX #$FF
140     BEQ EXIT
150     JMP LOOP
160 EXIT
170     LDA #$3
180     STA $9B08
190     LDA #$98
200     STA $2F4
210     RTS
```

[תשובה 22 מעמ' 95]

```
10 C=0

20 READ D:IF D=-1 THEN 50

30 POKE 1536+C,D

40 C=C+1:GOTO 20

50 C=0:RESTORE 200

60 READ D:IF D=-1 THEN 90

70 POKE 1616+C,D

80 C=C+1:GOTO 60

90 U=USR(1536)

100 POKE 548,80:POKE 549,6

110 ? "aaaaaaa"

120 ? "aHELLOa"

130 ? "aaaaaaa"

140 DATA 104,162,0,232,189,0,224,157,0,
152,189,0,225,157,0,153,189,0,226,157,
0,154,189,0,227,157,0,155,224

150 DATA 255,240,3,76,3,6,169,3,141,8,
155,169,152,141,244,2,96,-1

200 DATA 8,72,165,20,41,7,201,7,208,44,
173,8,155,201,1,240,42,201,128,240,38,
24,42,141,8,155,42,141,9,155

210 DATA 42,141,10,155,42,141,11,155,
42,141,12,155,42,141,13,155,42,141,14,
155,42,141,15,155,104

220 DATA 40,76,98,228,56,76,102,6,-1
```

ביחידת לימוד זו תכיר את אופן פעולתו של ה-Antic (השבב הגרפי של מחשב האטארי) ותלמד לבנות בעצמך מסכים גרפיים עשירים ומורכבים ואף לגלול אותם לאורך ולרוחב.

בנוסף תלמד לשלוט ב"צינור" המזרים מידע גרפי בין ה-Antic לבין מסך הטלוויזיה שלך ולהפעיל על אותו "צינור" מניפולציות שונות כדי ליצור אפקטים מרהיבים.

נכתב ע"י: נדב שטכמן פולישוק